Chapter 8
Spatial Discretization: The Diffusion Term

Abstract This chapter describes in detail the discretization of the diffusion term represented by the spatial Laplacian operator. It is investigated separately from the convection term, because convection and diffusion represent two distinct physical phenomena. Thus from a numerical point of view, they have to be handled differently, requiring distinct interpolation profiles with disparate considerations. The chapter begins with the discretization of the diffusion equation in the presence of a source term over a two-dimensional rectangular domain using a Cartesian grid system. The adopted interpolation profile for the variation of the dependent variable between grid points and the basic rules that should be satisfied by the coefficients of the discretized equation are discussed. The chapter proceeds with a discussion on the implementation of the Dirichlet, Von Neumann, mixed, and symmetry boundary conditions. The discretization over a non-Cartesian orthogonal grid is then introduced, followed by a detailed description of the discretization on non-orthogonal structured and unstructured grid systems. The treatment of the non-orthogonal cross-diffusion contribution, which necessitates computation of the gradient, is clarified. Then anisotropic diffusion is introduced and handled following the same methodology developed for isotropic diffusion. The under-relaxation procedure needed for highly non-linear problems is outlined. The chapter ends with computational pointers explaining the treatment of diffusion in both uFVM and OpenFOAM®.

8.1 Two-Dimensional Diffusion in a Rectangular Domain

A simple rectangular domain with a regular Cartesian grid, as shown in Fig. 8.1, is first considered. The aim is to discretize, on this domain, the steady-state diffusion equation given by

$$-\nabla \cdot \left(\Gamma^{\phi} \nabla \phi\right) = Q^{\phi} \tag{8.1}$$

F. Moukalled et al., *The Finite Volume Method in Computational Fluid Dynamics*,
Fluid Mechanics and Its Applications 113, DOI 10.1007/978-3-319-16874-6_8

where ϕ denotes a scalar variable (e.g., temperature, mass fraction of a chemical species, turbulence kinetic energy, etc.), Q^{ϕ} the generation of ϕ per unit volume within the domain, and Γ^{ϕ} the diffusion coefficient. The equation can be written more generally in term of a diffusion flux $\mathbf{J}^{\phi,D}$ as

$$\nabla \cdot \mathbf{J}^{\phi,D} = Q^{\phi} \tag{8.2}$$

where $\mathbf{J}^{\phi,D}$ is defined as

$$\mathbf{J}^{\phi,D} = -\Gamma^{\phi} \nabla \phi \tag{8.3}$$

Following the first stage discretization presented in Chap. 5, Eq. (8.1) can be formulated as

$$\sum_{f \sim nb(C)} \left(-\Gamma^{\phi} \nabla \phi\right)_f \cdot \mathbf{S}_f = Q_C^{\phi} V_C \tag{8.4}$$

The expanded form of the above equation can be written as

$$\left(-\Gamma^{\phi} \nabla \phi\right)_e \cdot \mathbf{S}_e + \left(-\Gamma^{\phi} \nabla \phi\right)_w \cdot \mathbf{S}_w + \left(-\Gamma^{\phi} \nabla \phi\right)_n \cdot \mathbf{S}_n + \left(-\Gamma^{\phi} \nabla \phi\right)_s \cdot \mathbf{S}_s = Q_C^{\phi} V_C \tag{8.5}$$

For the uniform Cartesian grid of Fig. 8.1, the surface vectors normal to the element faces are given by

$$\begin{aligned} \mathbf{S}_e &= +(\Delta y)_e \, \mathbf{i} = \|\mathbf{S}_e\| \mathbf{i} = S_e \mathbf{i} \quad \mathbf{S}_w = -(\Delta y)_w \, \mathbf{i} = -\|\mathbf{S}_w\| \mathbf{i} = -S_w \mathbf{i} \\ \mathbf{S}_n &= +(\Delta x)_n \, \mathbf{j} = \|\mathbf{S}_n\| \mathbf{j} = S_n \mathbf{j} \quad \mathbf{S}_s = -(\Delta x)_s \, \mathbf{j} = -\|\mathbf{S}_s\| \mathbf{j} = -S_s \mathbf{j} \end{aligned} \tag{8.6}$$

Thus for the east face the diffusion flux is found to be

$$\begin{aligned} J_e^{\phi,D} &= -\left(\Gamma^{\phi} \nabla \phi\right)_e \cdot \mathbf{S}_e \\ &= -\Gamma_e^{\phi} S_e \left(\frac{\partial \phi}{\partial x} \mathbf{i} + \frac{\partial \phi}{\partial y} \mathbf{j} \right)_e \cdot \mathbf{i} \\ &= -\Gamma_e^{\phi} (\Delta y)_e \left(\frac{\partial \phi}{\partial x} \right)_e \end{aligned} \tag{8.7}$$

Following the discrete conservation equation for one integration point discussed earlier, the discrete form of the diffusion flux can be written as

$$J_e^{\phi,D} = FluxT_e = FluxC_e \phi_C + FluxF_e \phi_F + FluxV_e \tag{8.8}$$

To determine the $FluxC_e$, $FluxF_e$, and $FluxV_e$ coefficients, a profile describing the variation of ϕ between the centroids of the two elements sharing the face, where the gradient has to be computed, is required. Assuming that ϕ varies linearly

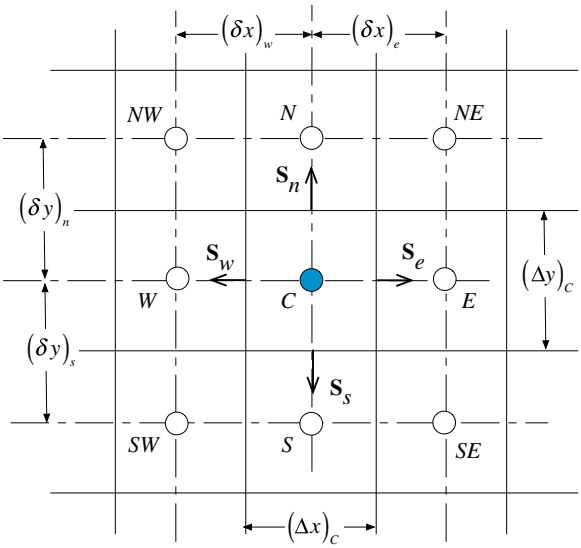

Fig. 8.1 A uniform Cartesian grid

between cell centroids (Fig. 8.2), the gradient at face e along the **i** direction may be written as

$$\left(\frac{\partial \phi}{\partial x}\right)_e = \frac{\phi_E - \phi_C}{(\delta x)_e} \tag{8.9}$$

Substituting into Eq. (8.8), the discretized form of the diffusion flux along face e is obtained as

$$\begin{aligned} FluxT_e &= -\Gamma_e^{\phi}(\Delta y)_e \frac{(\phi_E - \phi_C)}{\delta x_e} \\ &= \Gamma_e^{\phi} \frac{(\Delta y)_e}{\delta x_e}(\phi_C - \phi_E) \\ &= FluxC_e\phi_C + FluxF_e\phi_E + FluxV_e \end{aligned} \tag{8.10}$$

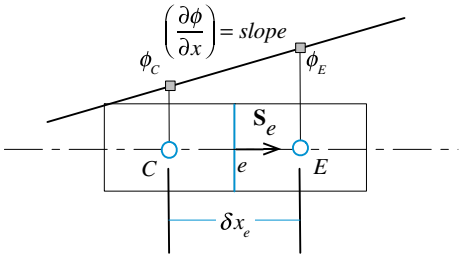

Fig. 8.2 Interpolation profile at face e and slope of gradient

Taking

$$gDiff_e = \frac{(\Delta y)_e}{\delta x_e} = \frac{\|\mathbf{S}_e\|}{\|\mathbf{d}_{CE}\|} = \frac{S_e}{d_{CE}} \tag{8.11}$$

where $\mathbf{d}_{CE}$ is the distance vector between the centroids of elements C and E, the coefficients become

$$\begin{aligned} FluxC_e &= \Gamma_e^{\phi} gDiff_e \\ FluxF_e &= -\Gamma_e^{\phi} gDiff_e \\ FluxV_e &= 0 \end{aligned} \tag{8.12}$$

A similar procedure applied to face w yields

$$\begin{aligned} FluxT_w &= -\left(\Gamma^{\phi}\nabla\phi\right)_w \cdot \mathbf{S}_w \\ &= -\Gamma_w^{\phi} S_w \left(\frac{\partial\phi}{\partial x}\mathbf{i} + \frac{\partial\phi}{\partial y}\mathbf{j}\right)_w \cdot (-\mathbf{i}) \\ &= \Gamma_w^{\phi} S_w \left(\frac{\partial\phi}{\partial x}\right)_w \\ &= \Gamma_w^{\phi} S_w \frac{(\phi_C - \phi_W)}{(\delta x)_w} \\ &= FluxC_w\phi_C + FluxF_w\phi_W + FluxV_w \end{aligned} \tag{8.13}$$

where now

$$\begin{aligned} FluxC_w &= \Gamma_w^{\phi} gDiff_w \\ FluxF_w &= -\Gamma_w^{\phi} gDiff_w \\ FluxV_w &= 0 \end{aligned} \tag{8.14}$$

and

$$gDiff_w = \frac{(\Delta y)_w}{\delta x_w} = \frac{\|\mathbf{S}_w\|}{\|\mathbf{d}_{CW}\|} = \frac{S_w}{d_{cw}}$$

Similar expressions may be written along faces n and s and are given by

$$\begin{aligned} FluxT_n &= FluxC_n\phi_C + FluxF_n\phi_N + FluxV_n \\ FluxT_s &= FluxC_s\phi_C + FluxF_s\phi_S + FluxV_s \end{aligned} \tag{8.15}$$

where

$$\begin{aligned} FluxC_n &= \Gamma_n^\phi gDiff_n \quad FluxF_n = -\Gamma_n^\phi gDiff_n \quad FluxV_n = 0 \\ FluxC_s &= \Gamma_s^\phi gDiff_s \quad FluxF_s = -\Gamma_s^\phi gDiff_s \quad FluxV_s = 0 \end{aligned} \tag{8.16}$$

and

$$\begin{aligned} gDiff_n &= \frac{\|\mathbf{S}_n\|}{\|\mathbf{d}_{CN}\|} = \frac{(\Delta x)_n}{\delta y_n} = \frac{S_n}{d_{CN}} \\ gDiff_s &= \frac{\|\mathbf{S}_s\|}{\|\mathbf{d}_{CS}\|} = \frac{(\Delta x)_s}{\delta y_s} = \frac{S_s}{d_{CS}} \end{aligned} \tag{8.17}$$

Substituting into Eq. (8.5), the algebraic form of the diffusion equation is obtained as

$$a_C\phi_C + a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S = b_C \tag{8.18}$$

where

$$\begin{aligned} a_E &= FluxF_e = -\Gamma_e^\phi gDiff_e \\ a_W &= FluxF_w = -\Gamma_w^\phi gDiff_w \\ a_N &= FluxF_n = -\Gamma_n^\phi gDiff_n \\ a_S &= FluxF_s = -\Gamma_s^\phi gDiff_s \\ a_C &= FluxC_e + FluxC_w + FluxC_n + FluxC_s \\ &= -(a_E + a_W + a_N + a_S) \\ b_C &= Q_C^\phi V_C - (FluxV_e + FluxV_w + FluxV_n + FluxV_s) \end{aligned} \tag{8.19}$$

or, more compactly, as

$$a_C\phi_C + \sum_{F\sim NB(C)} a_F\phi_F = b_C \tag{8.20}$$

with

$$\begin{aligned} a_F &= FluxF_f = -\Gamma_f^\phi gDiff_f \\ a_C &= \sum_{f\sim nb(C)} FluxC_f \\ b_C &= Q_C^\phi V_C - \sum_{f\sim nb(C)} FluxV_f \end{aligned} \tag{8.21}$$

where the subscript F denotes the neighbors of element C (E, W, N, S), and the subscript f denotes the neighboring faces of element C (e, w, n, s).

8.2 Comments on the Discretized Equation

A proper discretization method should result in a discretized algebraic equation that reflects the characteristics of the original conservation equation. The properties of the discretization techniques were introduced in the previous chapter and two additional rules that the coefficients of the discretized equation have to satisfy are presented next.

8.2.1 The Zero Sum Rule

Looking back at the discretization process, the first major approximation made was the linear profile assumption for the variation of ϕ between the centroids of the elements straddling the element face. The reader might ask why to use a first order rather than a higher order profile. To answer this question, a one dimensional configuration with no source term is considered. Under these conditions the discretized equation reduces to

$$a_C\phi_C + a_E\phi_E + a_W\phi_W = 0 \tag{8.22}$$

where

$$a_E = -\Gamma_e^{\phi} gDiff_e \quad a_W = -\Gamma_w^{\phi} gDiff_w \quad a_C = -(a_E + a_W) \tag{8.23}$$

In the absence of any source or sink within this one dimensional domain, the transfer of ϕ occurs by diffusion only and is governed by Fourier's law (elliptic equation), i.e., in the direction of decreasing ϕ. As such, the value of ϕ_e or ϕ_w should lie between the values ϕ_C and ϕ_E or ϕ_C and ϕ_W, respectively, which is guaranteed by the linear profile. A second order profile (e.g., a parabolic profile) may result in a value at the face that is higher or lower than the values at the centroids of the cells straddling the face, which is unphysical. The same is true for other higher order profiles. If the discretization scheme is to guarantee physical results, then a linear profile should be used. Moreover, the adopted profile becomes less important as the size of the element decreases, since all approximations are expected to yield the same analytical solution in the limit when the size of the element approaches zero. Furthermore, in the absence of any source term, the multi-dimensional heat conduction equation reduces to,

$$-\nabla \cdot \left(\Gamma^{\phi}\nabla\phi\right) = 0 \tag{8.24}$$

This implies that ϕ and ϕ + *constant* are solutions to the conservation equation. A consistent discretization method should reflect this property through its discretized equation and the discretized equation should fulfill

$$\left.\begin{aligned} a_C\phi_C + \sum_{F\sim NB(C)} a_F\phi_F = 0 \\ a_C(\phi_C + constant) + \sum_{F\sim NB(C)} a_F(\phi_F + constant) = 0 \end{aligned}\right\} \Rightarrow a_C + \sum_{F\sim NB(C)} a_F = 0 \tag{8.25}$$

which is actually satisfied by the discretized equation. The above equation, which is valid in the presence or absence of a source/sink term as revealed by Eq. (8.19), may be written as

$$a_C = -\sum_{F\sim NB(C)} a_F \tag{8.26}$$

or as

$$\sum_{F\sim NB(C)} \frac{a_F}{a_C} = -1 \tag{8.27}$$

Thus ϕ_C can be viewed as the weighted sum of its neighbors, and in the absence of any source term it should always be bounded by these neighboring ϕ_F values. When a source term is present, i.e., when $S_C^\phi \neq 0$, ϕ_C does not need to be bounded in this manner, and can over/undershoot the neighboring values, but this is perfectly physical. The extent of over/under shoot is determined by the magnitude of S_C^ϕ with respect to the size of the coefficients at the neighboring nodes (a_F).

8.2.2 The Opposite Signs Rule

The above derivations demonstrated that the coefficients a_C and a_F are of opposite signs. This is of physical significance implying that as the value of ϕ_F is increased/decreased, the value of ϕ_C is expected to increase/decrease. This is basically related to boundedness suggesting that a sufficient condition for this property to be satisfied is for the neighboring and main coefficients to be of opposite signs. If not, then the boundedness property may not be enforced.

8.3 Boundary Conditions

It is well known that the analytical solution to any ordinary or partial differential equation is obtained up to some constants that are fixed by the applicable boundary conditions to the situation being studied. Therefore, using different boundary

conditions will result in different solutions even though the general equation remains the same. Numerical solutions follow the same constrain, necessitating correct and accurate implementation of boundary conditions as any slight change in these conditions introduced by the numerical approximation leads to a wrong solution of the problem under consideration.

Boundary conditions will be discussed as deemed relevant to the terms being discretized. For conduction/diffusion problems Dirichlet, Neumann, mixed, and symmetry boundary condition types are encountered, which are detailed next.

Boundary conditions are applied on boundary elements, which have one or more faces on the boundary. Discrete values of ϕ are stored both at centroids of boundary cells and at centroids of boundary faces.

Let C denotes the centroid of the boundary element shown in Fig. 8.3 with one boundary face of centroid b and of surface vector $\mathbf{S}_b$ pointing outward. As before, the discretization process over cell C yields

$$\sum_{f\sim nb(C)} \left(\mathbf{J}^{\phi,D}\cdot\mathbf{S}\right)_f = Q_C^{\phi} V_C \tag{8.28}$$

The fluxes on the interior faces are discretized as before, while the boundary flux is discretized with the aim of constructing a linearization with respect to ϕ_C, thus

$$\begin{aligned}\mathbf{J}_b^{\phi,D}\cdot\mathbf{S}_b &= FluxT_b \\ &= -\Gamma_b^{\phi}(\nabla\phi)_b\cdot\mathbf{S}_b \\ &= FluxC_b\phi_C + FluxV_b\end{aligned} \tag{8.29}$$

The specification of boundary conditions involves either specifying the unknown boundary value ϕ_b, or alternatively, the boundary flux $\mathbf{J}_b^{\phi,D}$. Using Eq. (8.18), the discretized equation at a boundary element for the different boundary condition types of diffusion problems are derived next.

8.3.1 Dirichlet Boundary Condition

A Dirichlet boundary condition is a type of boundary condition that specifies the value of ϕ at the boundary, i.e.,

$$\phi_b = \phi_{specified} \tag{8.30}$$

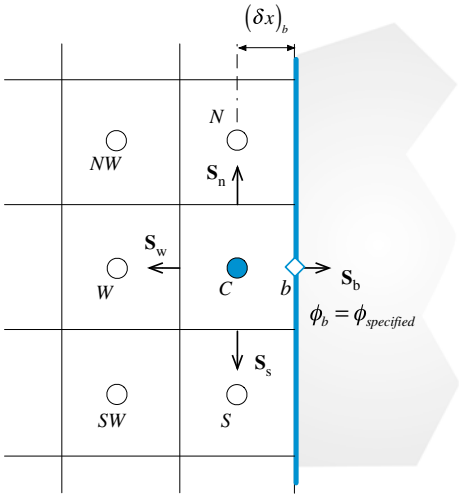

Fig. 8.3 Value specified boundary condition

For this case

$$\begin{aligned} FluxT_b &= -\Gamma_b^{\phi}(\nabla\phi)_b \cdot \mathbf{S}_b \\ &= -\Gamma_b^{\phi}\frac{\|\mathbf{S}_b\|}{\|\mathbf{d}_{Cb}\|}(\phi_b - \phi_C) \\ &= FluxC_b\phi_C + FluxV_b \end{aligned} \tag{8.31}$$

yielding

$$\begin{aligned} FluxC_b &= \Gamma_b^{\phi} gDiff_b = a_b \\ FluxV_b &= -\Gamma_b^{\phi} gDiff_b \phi_b = -a_b\phi_b \end{aligned} \tag{8.32}$$

with

$$gDiff_b = \frac{S_b}{d_{Cb}} \tag{8.33}$$

Thus for element C shown in Fig. 8.3 the a_E coefficient is zero reducing the discretized equation to

$$a_C\phi_C + a_W\phi_W + a_N\phi_N + a_S\phi_S = b_C \tag{8.34}$$

where

$$\begin{aligned} a_E &= 0 \\ a_W &= FluxF_w = -\Gamma_w^{\phi} gDiff_w \\ a_N &= FluxF_n = -\Gamma_n^{\phi} gDiff_n \\ a_S &= FluxF_s = -\Gamma_s^{\phi} gDiff_s \\ a_C &= FluxC_b + \sum_{f\sim nb(C)} FluxC_f = FluxC_b + (FluxC_w + FluxC_n + FluxC_s) \end{aligned} \tag{8.35}$$

and

$$b_C = Q_C^{\phi} V_C - \left(FluxV_b + \sum_{f \sim nb(C)} FluxV_f \right) \tag{8.36}$$

The following important observations can be made about the discretized boundary equation:

1. The coefficient a_b is larger than other neighbor coefficients because b is closer to C and consequently has a more important effect on ϕ_C.
2. The coefficient a_C is still the sum of all neighboring coefficients including a_b. This means that for the boundary element $\sum_{F \sim NB(C)} |a_F|/|a_C| < 1$ giving the second necessary condition to satisfy the Scarborough criterion, thus guaranteeing, at any one iteration, the convergence of the linear system of equations via an iterative solution method.
3. The $a_b\phi_b$ product (=$FluxV_b$) is now on the right hand side of the equation, i.e., part of b_C, because it contains no unknowns.

8.3.2 Von Neumann Boundary Condition

If the flux (or normal gradient to the face) of ϕ is specified at the boundary (Fig. 8.4), then the boundary condition is denoted by a Neumann boundary condition. In this case the specified flux is given by

$$-\left(\Gamma^{\phi} \nabla \phi\right)_b \cdot \mathbf{i} = q_b \tag{8.37}$$

which is, in effect, the flux $\mathbf{J}_b^{\phi,D}$, since

$$\begin{aligned} \mathbf{J}_b^{\phi,D} \cdot \mathbf{S}_b &= -\left(\Gamma^{\phi} \nabla \phi\right)_b \cdot \|\mathbf{S}_b\| \mathbf{i} = q_b \|\mathbf{S}_b\| \\ &= FluxC_b \phi_C + FluxV_b \end{aligned} \tag{8.38}$$

where now

$$\begin{aligned} FluxC_b &= 0 \\ FluxV_b &= q_b S_b \\ &= q_b (\Delta y)_C \end{aligned} \tag{8.39}$$

Here the flux components are assumed to be positive when they act in the same direction as the coordinate system used.

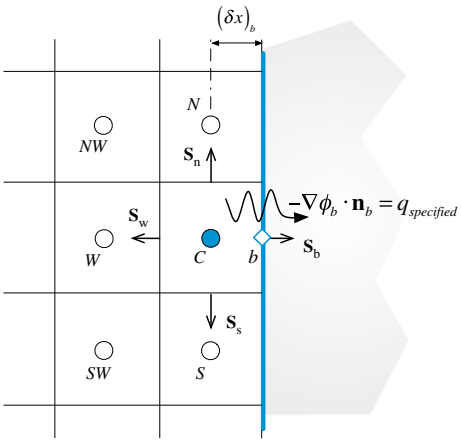

Fig. 8.4 Flux specified boundary condition

Thus q_b may directly be included in Eq. (8.18) to yield the following discrete equation for the boundary cell C:

$$a_C\phi_C + a_W\phi_W + a_N\phi_N + a_S\phi_S = b_C \tag{8.40}$$

where

$$\begin{aligned}
a_E &= 0 \\
a_W &= FluxF_w = -\Gamma_w^\phi gDiff_w \\
a_N &= FluxF_n = -\Gamma_n^\phi gDiff_n \\
a_S &= FluxF_s = -\Gamma_s^\phi gDiff_s \\
a_C &= \sum_{f\sim nb(C)} FluxC_f = -(a_W + a_N + a_S) \\
b_C &= Q_C^\phi V_C - \left(FluxV_b + \sum_{f\sim nb(C)} FluxV_f \right)
\end{aligned} \tag{8.41}$$

The following important points can be made about the above discretized equation:

1. A Von Neumann boundary condition does not result in a dominant a_C coefficient.
2. If both q_b and S_C^ϕ are zero, then ϕ_C will be bounded by its neighbors. Otherwise, ϕ_C can exceed (or fall below) the neighbor values of ϕ, which is admissible. If ϕ is temperature, for example, then q_b represents the heat flux applied at the boundary. Therefore if heat is added at the boundary, then the temperature in the region close to the boundary is expected to be higher than that in the interior.

3. Once ϕ_C is computed, the boundary value ϕ_b may be computed using

$$\phi_b = \frac{\Gamma_b^{\phi} gDiff_b \phi_C - q_b}{\Gamma_b^{\phi} gDiff_b} \tag{8.42}$$

4. Finally the Von Neumann condition can be considered as a natural boundary condition for the Finite Volume method since, for the case where the specified flux is zero, nothing needs to be done in terms of discretization for the face, while a specified value of zero (Dirichlet condition) will still require the discretization to be carried out.

8.3.3 Mixed Boundary Condition

The mixed boundary condition, schematically depicted in Fig. 8.5, refers to the situation where information at the boundary is given via a convection transfer coefficient (h_∞) and a surrounding value for $\phi(\phi_\infty)$ as

$$\mathbf{J}_b^{\phi,D} \cdot \mathbf{S}_b = -\left(\Gamma^{\phi} \nabla \phi\right)_b \cdot \mathbf{i} S_b = -h_\infty (\phi_\infty - \phi_b)(\Delta y)_C \tag{8.43}$$

which can be rewritten as

$$-\Gamma_b^{\phi} S_b \left(\frac{\phi_b - \phi_C}{\delta x_b}\right) = -h_\infty (\phi_\infty - \phi_b) S_b \tag{8.44}$$

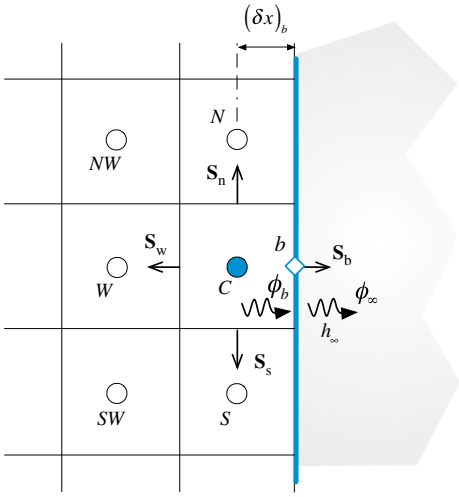

Fig. 8.5 Mixed type boundary condition

from which an equation for ϕ_b is obtained as

$$\phi_b = \frac{h_\infty \phi_\infty + \left(\Gamma_b^\phi / \delta x_b\right)\phi_C}{h_\infty + \left(\Gamma_b^\phi / \delta x_b\right)} \tag{8.45}$$

Substituting ϕ_b back in Eq. (8.43), the flux equation is transformed to

$$\begin{aligned} \mathbf{J}_b^{\phi,D} \cdot \mathbf{S}_b &= -\underbrace{\left[\frac{h_\infty\left(\Gamma_b^\phi/\delta x_b\right)}{h_\infty + \left(\Gamma_b^\phi/\delta x_b\right)} S_b\right]}_{R_{eq}} (\phi_\infty - \phi_C) \\ &= FluxC_b \phi_C + FluxV_b \end{aligned} \tag{8.46}$$

where now

$$\begin{aligned} FluxC_b &= R_{eq} \\ FluxV_b &= -R_{eq}\phi_\infty \end{aligned} \tag{8.47}$$

Using the flux term given by Eq. (8.47) in the discretized equation of the boundary element C, the modified equation is found to be

$$a_C \phi_C + a_W \phi_W + a_N \phi_N + a_S \phi_S = b_C \tag{8.48}$$

where

$$\begin{aligned} a_E &= 0 \\ a_W &= FluxF_w = -\Gamma_w^\phi gDiff_w \\ a_N &= FluxF_n = -\Gamma_n^\phi gDiff_n \\ a_S &= FluxF_s = -\Gamma_s^\phi gDiff_s \\ a_C &= FluxC_b + \sum_{f\sim nb(C)} FluxC_f = (FluxC_b + FluxC_w + FluxC_n + FluxC_s) \\ b_C &= Q_C^\phi V_C - \left(FluxV_b + \sum_{f=nb(C)} FluxV_f\right) \end{aligned} \tag{8.49}$$

8.3.4 Symmetry Boundary Condition

Along a symmetry boundary the normal flux to the boundary of a scalar variable ϕ is zero. Therefore a symmetry boundary condition is equivalent to a Neumann boundary condition with the value of the flux set to zero (i.e., $FluxC_b = FluxV_b = 0$).

Thus the modified equation along a symmetry boundary condition can be deduced from Eqs. (8.40) and (8.41) by setting q_b to zero and is given by

$$a_C\phi_C + a_W\phi_W + a_N\phi_N + a_S\phi_S = b_C \tag{8.50}$$

where

$$\begin{aligned} a_E &= 0 \\ a_W &= FluxF_w = -\Gamma_w^\phi gDiff_w \\ a_N &= FluxF_n = -\Gamma_n^\phi gDiff_n \\ a_S &= FluxF_s = -\Gamma_s^\phi gDiff_s \\ a_C &= \sum_{f\sim nb(C)} FluxC_f = -(a_W + a_N + a_S) \\ b_C &= Q_C^\phi V_C - \sum_{f\sim nb(C)} FluxV_f \end{aligned} \tag{8.51}$$

8.4 The Interface Diffusivity

In the above discretized equation, the diffusion coefficients Γ_e^ϕ, Γ_w^ϕ, Γ_n^ϕ, and Γ_s^ϕ have been used to represent the value of Γ^ϕ at the e, w, n and s faces of the element, respectively. When the diffusion coefficient Γ^ϕ varies with position, its value will be known at the cell centroids E, W, … and so on. Then a prescription for evaluating the interface value is needed in terms of values at these grid points. The following discussion is, of course, not relevant to situations of uniform diffusion coefficient.

The discussion may become clearer if the energy equation is considered, in which case the diffusion coefficient represents the conductivity of the material used and ϕ denotes the temperature. Non-uniform conductivity occurs in non-homogeneous materials and/or when the thermal conductivity is temperature dependent. In the treatment of the general differential equation for ϕ, the diffusion coefficient Γ^ϕ will be handled in the same way. Significant variations of Γ^ϕ are frequently encountered for example, in turbulent flow, where Γ^ϕ may stand for the turbulent viscosity or the turbulent conductivity. Thus, a proper formulation for non-uniform Γ^ϕ is highly desirable.

A simple approach for calculating the interface conductivity is the linear profile assumption for the variation of Γ^ϕ, between point C and any of its neighbors. Thus, along the east face the value of Γ^ϕ, is obtained as

$$\Gamma_e^\phi = (1 - g_e)\Gamma_C^\phi + g_e\Gamma_E^\phi \tag{8.52}$$

where the interpolation factor g_e is a ratio in terms of distances between centroids given by

$$g_e = \frac{d_{Ce}}{d_{Ce} + d_{eE}} \tag{8.53}$$

Hence for a Cartesian grid if the interface is midway between the grid points, g_e would be 0.5, and Γ_e^ϕ would be the arithmetic mean of Γ_C^ϕ and Γ_E^ϕ. Similar coefficients can be defined for other faces as

$$\begin{aligned} g_w &= \frac{d_{Cw}}{d_{Cw} + d_{wW}} \\ g_n &= \frac{d_{Cn}}{d_{Cn} + d_{nN}} \\ g_s &= \frac{d_{Cs}}{d_{Cs} + d_{sS}} \end{aligned} \tag{8.54}$$

Therefore it is sufficient to calculate the coefficients of each surface of an element only once. Moreover, the use of distances instead of volumes will lead to the same interpolation factors as the grid here is Cartesian.

This basic approach leads to rather incorrect implications in some cases and cannot accurately handles, for example, abrupt changes of conductivity that may occur in composite materials. Fortunately, a much better alternative of comparable simplicity is available. In developing this alternative, it is recognized that the local value of conductivity at an interface is not of primary concern. Rather, the main objective is to obtain a good representation of the diffusion flux $\mathbf{J}^{\phi,D}$ at the interface [1].

For the one dimensional problem shown in Fig. 8.6, it is assumed that the element C is composed of a material having a thermal conductivity Γ_C^ϕ, while element E is made of a material of thermal conductivity Γ_E^ϕ. For the non-homogeneous slab between points C and E, a steady one-dimensional analysis (without sources) results in (the flux on either side of the interface e is supposed to be the same)

$$\mathbf{J}_e^{\phi,D} \cdot \mathbf{S}_e = \frac{\phi_C - \phi_e}{\frac{(\delta x)_{Ce}}{\Gamma_C^\phi}} = \frac{\phi_e - \phi_E}{\frac{(\delta x)_{eE}}{\Gamma_E^\phi}} = \frac{\phi_C - \phi_E}{\frac{(\delta x)_{Ce}}{\Gamma_C^\phi} + \frac{(\delta x)_{eE}}{\Gamma_E^\phi}} = \frac{\phi_C - \phi_E}{\frac{(\delta x)_{CE}}{\Gamma_e^\phi}} \tag{8.55}$$

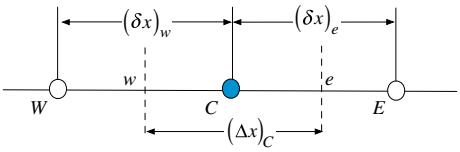

Fig. 8.6 Interpolation of properties at element faces

Hence the effective conductivity for the slab is found to be

$$\frac{(\delta x)_{CE}}{\Gamma_e^\phi} = \frac{(\delta x)_{Ce}}{\Gamma_C^\phi} + \frac{(\delta x)_{eE}}{\Gamma_E^\phi} \Rightarrow \frac{1}{\Gamma_e^\phi} = \left(\frac{1 - g_e}{\Gamma_E^\phi} + \frac{g_e}{\Gamma_C^\phi} \right) \tag{8.56}$$

When the interface is halfway between C and E ($g_e = 0.5$), Eq. (8.56) reduces to

$$\Gamma_e^\phi = \frac{2\Gamma_C^\phi \Gamma_E^\phi}{\Gamma_C^\phi + \Gamma_E^\phi} \tag{8.57}$$

which is the ***harmonic mean*** of Γ_C^ϕ and Γ_E^ϕ, rather than the ***arithmetic*** mean.

It is important to note that the harmonic mean interpolation for discontinuous diffusion coefficients is exact only for one-dimensional diffusion. Nevertheless, its application for multi-dimensional situation has an important advantage. With this type of interpolation, nothing special need to be done when treating conjugate interfaces. Solid and fluid cells are simply treated as part of the same domain with different diffusion coefficients stored at the cell centroids. By calculating the face diffusivity as the harmonic-mean of the values at the centroids sharing the face, the diffusion flux at the conjugate interface is correctly computed.

Example 1

The heat conduction in the two-dimensional rectangular domain composed of two materials shown in Fig. 8.7 *is governed by the following differential equation*:

$$\nabla \cdot (k \nabla T) = 0$$

where T represents temperature. For the thermal conductivities (k) and boundary conditions displayed in the figure:

(a) *Derive the algebraic equations for all the elements shown in the figure.*
(b) *Using the Gauss-Seidel iterative method discussed in Chap. 4, solve the system of equations obtained and compute the cell values of T.*
(c) *Compute the values of T at the bottom, right, and top boundaries.*
(d) *Compute the net heat transfer through the top, bottom, and left boundaries and check that energy conservation is satisfied.*

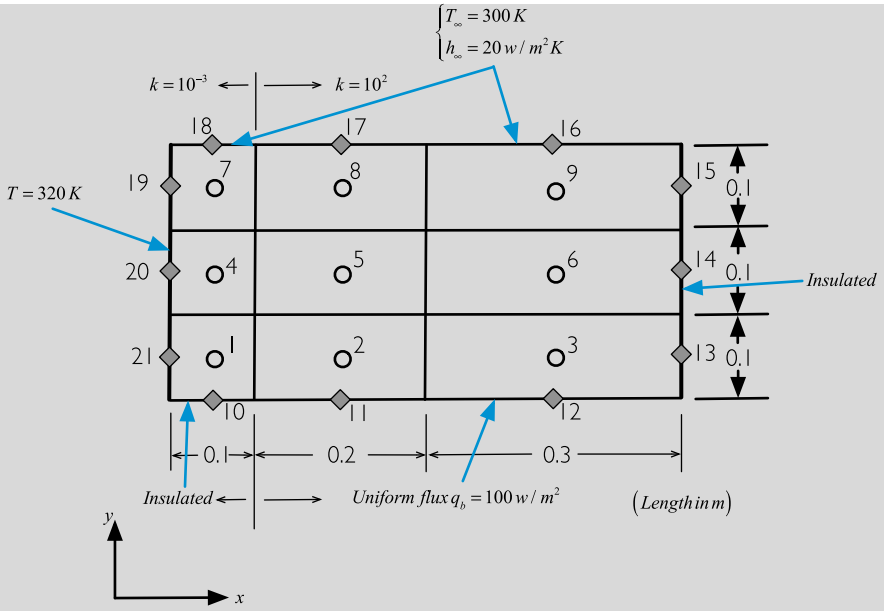

Fig. 8.7 Conduction heat transfer in a two dimensional rectangular domain

Solution

The first step is to determine the geometric quantities that are needed in the solution. The coordinates of the various nodes are found to be

$$x_{21} = x_{20} = x_{19} = 0 \qquad y_{10} = y_{11} = y_{12} = 0$$
$$x_{10} = x_1 = x_4 = x_7 = x_{18} = 0.05 \qquad y_{21} = y_1 = y_2 = y_3 = y_{13} = 0.05$$
$$x_{11} = x_2 = x_5 = x_8 = x_{17} = 0.2 \qquad y_{20} = y_4 = y_5 = y_6 = y_{14} = 0.15$$
$$x_{12} = x_3 = x_6 = x_9 = x_{16} = 0.45 \qquad y_{19} = y_7 = y_8 = y_9 = y_{15} = 0.25$$
$$x_{13} = x_{14} = x_{15} = 0.6 \qquad y_{18} = y_{17} = y_{16} = 0.3$$

The distance between nodes are computed as

$$\delta x_{21-1} = \delta x_{20-4} = \delta x_{19-7} = 0.05 \qquad \delta y_{10-1} = \delta y_{11-2} = \delta y_{12-3} = 0.05$$
$$\delta x_{1-2} = \delta x_{4-5} = \delta x_{7-8} = 0.15 \qquad \delta y_{1-4} = \delta y_{2-5} = \delta y_{3-6} = 0.1$$
$$\delta x_{2-3} = \delta x_{5-6} = \delta x_{8-9} = 0.25 \qquad \delta y_{4-7} = \delta y_{5-8} = \delta y_{6-9} = 0.1$$
$$\delta x_{3-13} = \delta x_{6-14} = \delta x_{9-15} = 0.15 \qquad \delta y_{7-18} = \delta y_{8-17} = \delta y_{9-16} = 0.05$$

The size of the elements are given by

$$\begin{array}{ll} \Delta x_1 = \Delta x_4 = \Delta x_7 = 0.1 & \Delta y_1 = \Delta y_2 = \Delta y_3 = 0.1 \\ \Delta x_2 = \Delta x_5 = \Delta x_8 = 0.2 & \Delta y_4 = \Delta y_5 = \Delta y_6 = 0.1 \\ \Delta x_3 = \Delta x_6 = \Delta x_9 = 0.3 & \Delta y_7 = \Delta y_8 = \Delta y_9 = 0.1 \end{array}$$

The computed volumes of the various cells are obtained as

$$V_1 = V_4 = V_7 = 0.01 \quad V_2 = V_5 = V_8 = 0.02 \quad V_3 = V_6 = V_9 = 0.03$$

The interpolation factors needed to find values at element faces are calculated as

$$\begin{aligned} (g_e)_1 = (g_e)_4 = (g_e)_7 &= \frac{V_1}{V_1 + V_2} = \frac{0.01}{0.01 + 0.02} = 0.333 \\ (g_e)_2 = (g_e)_5 = (g_e)_8 &= \frac{V_2}{V_2 + V_3} = \frac{0.02}{0.02 + 0.03} = 0.4 \\ (g_n)_1 = (g_n)_2 = (g_n)_3 &= \frac{V_1}{V_1 + V_4} = \frac{0.01}{0.01 + 0.01} = 0.5 \\ (g_n)_4 = (g_n)_5 = (g_n)_6 &= \frac{V_{10}}{V_{10} + V_{15}} = \frac{0.01}{0.01 + 0.01} = 0.5 \end{aligned}$$

The thermal conductivity values over the domain are given by

$$k_1 = k_4 = k_7 = k_{10} = k_{21} = k_{20} = k_{19} = k_{18} = 10^{-3}$$
$$k_2 = k_3 = k_5 = k_6 = k_8 = k_9 = k_{11} = k_{12} = k_{13} = k_{14} = k_{15} = k_{16} = k_{17} = 10^2$$

while values at the element faces are found to be

$$k_{1-2} = \frac{k_1 k_2}{\left[1 - (g_e)_1\right] k_1 + (g_e)_1 k_2} = \frac{10^{-3} \times 10^2}{(1 - 0.333) \times 10^{-3} + 0.333 \times 10^{-2}} \approx 3 \times 10^{-3}$$
$$k_{1-2} = k_{4-5} = k_{7-8} = 3 \times 10^{-3} \quad k_{1-4} = k_{4-7} = 10^{-3} \quad k_{2-5} = k_{3-6} = k_{5-8} = k_{6-9} = 10^2$$

Using the above values, the discretized algebraic equations for all elements are derived next.

Element #1

The needed diffusion terms are calculated as

$$gDiff_e = \frac{\Delta y_1}{\delta x_{1-2}} = \frac{0.1}{0.15} = 0.667 \quad gDiff_w = \frac{\Delta y_1}{\delta x_{21-1}} = \frac{0.1}{0.05} = 2$$
$$gDiff_n = \frac{\Delta x_1}{\delta y_{1-4}} = \frac{0.1}{0.1} = 1$$

The interface conductivities are

$$k_e = k_{1-2} = 3 \times 10^{-3} \quad k_n = k_{1-4} = 10^{-3} \quad k_w = k_{21} = 10^{-3}$$

The general form of the equation is written as

$$a_C T_1 + a_E T_2 + a_N T_4 = b_C$$

where

$$a_E = FluxF_e = -k_e gDiff_e = -3 \times 10^{-3} \times 0.667 = -0.002$$
$$a_N = FluxF_n = -k_n gDiff_n = -10^{-3} \times 1 = -0.001$$

The west and south coefficients do not appear in the equation as their influence is integrated through the boundary conditions as

$$FluxC_w = k_w gDiff_w = 10^{-3} \times 2 = 0.002$$
$$FluxV_s = 0$$
$$FluxV_w = -k_w gDiff_w T_{21} = -10^{-3} \times 2 \times 320 = -0.64$$

The main coefficient and source term can now be calculated and are given by

$$a_C = FLuxC_e + FLuxC_n + FLuxC_w = 0.002 + 0.001 + 0.002 = 0.005$$
$$b_C = -FluxV_S - FluxV_w = 0 + 0.64 = 0.64$$

Substituting, the discretized algebraic equation is obtained as

$$0.005T_1 - 0.002T_2 - 0.001T_4 = 0.64$$

Element #2
The needed diffusion terms are calculated as

$$gDiff_e = \frac{\Delta y_2}{\delta x_{2-3}} = \frac{0.1}{0.25} = 0.4 \quad gDiff_w = \frac{\Delta y_2}{\delta x_{1-2}} = \frac{0.1}{0.15} = 0.667$$
$$gDiff_n = \frac{\Delta x_2}{\delta y_{2-5}} = \frac{0.2}{0.1} = 2$$

The interface conductivities are

$$k_e = k_{2-3} = 10^2 \quad k_n = k_{2-5} = 10^2 \quad k_w = k_{1-2} = 3 \times 10^{-3}$$

The general form of the equation is written as

$$a_C T_2 + a_E T_3 + a_W T_1 + a_N T_5 = b_C$$

where

$$a_E = FluxF_e = -k_e gDiff_e = -10^2 \times 0.4 = -40$$
$$a_W = FluxF_w = -k_w gDiff_w = -3 \times 10^{-3} \times 0.667 = -0.002$$
$$a_N = FluxF_n = -k_n gDiff_n = -10^2 \times 2 = -200$$

The south coefficient does not appear in the equation as its influence is integrated through the boundary conditions as

$$FluxV_s = q_b \mathbf{j} \cdot \mathbf{S}_1 = q_b \mathbf{j} \cdot (-\Delta x_2 \mathbf{j}) = -100 * 0.2 = -20$$

The main coefficient and source term can now be calculated and are given by

$$a_C = FLuxC_e + FLuxC_n + FLuxC_w = 40 + 0.002 + 200 = 240.002$$
$$b_C = -FluxV_s = 20$$

Substituting, the discretized algebraic equation is obtained as

$$240.002T_2 - 40T_3 - 0.002T_1 - 200T_5 = 20$$

Element #3
The needed diffusion terms are calculated as

$$gDiff_w = \frac{\Delta y_3}{\delta x_{2-3}} = \frac{0.1}{0.25} = 0.4 \quad gDiff_n = \frac{\Delta x_3}{\delta y_{3-6}} = \frac{0.3}{0.1} = 3$$

The interface conductivities are

$$k_w = k_{2-3} = 10^2 \quad k_n = k_{3-6} = 10^2$$

The general form of the equation is written as

$$a_C T_3 + a_W T_2 + a_N T_6 = b_C$$

where

$$a_W = FluxF_w = -k_w gDiff_w = -10^2 \times 0.4 = -40$$
$$a_N = FluxF_n = -k_n gDiff_n = -10^2 \times 3 = -300$$

The east and south coefficients do not appear in the equation as their influence is integrated through the boundary conditions as

$$FluxV_e = 0$$
$$FluxV_s = q_b\mathbf{j} \cdot \mathbf{S}_1 = q_b\mathbf{j} \cdot (-\Delta x_3\mathbf{j}) = -100 * 0.3 = -30$$

The main coefficient and source term can now be calculated and are given by

$$a_C = FLuxC_w + FLuxC_n = 40 + 300 = 340$$
$$b_C = -FluxV_S - FluxV_e = 30 + 0 = 30$$

Substituting, the discretized algebraic equation is obtained as

$$340T_3 - 40T_2 - 300T_6 = 30$$

Element #4
The needed diffusion terms are calculated as

$$gDiff_e = \frac{\Delta y_4}{\delta x_{4-5}} = \frac{0.1}{0.15} = 0.667 \quad gDiff_w = \frac{\Delta y_4}{\delta x_{20-4}} = \frac{0.1}{0.05} = 2$$
$$gDiff_n = \frac{\Delta x_4}{\delta y_{4-7}} = \frac{0.1}{0.1} = 1 \qquad gDiff_s = \frac{\Delta x_4}{\delta y_{1-4}} = \frac{0.1}{0.1} = 1$$

The interface conductivities are

$$k_e = k_{4-5} = 3 \times 10^{-3} \quad k_w = k_{20} = 10^{-3} \quad k_n = k_{4-7} = 10^{-3} \quad k_s = k_{1-4} = 10^{-3}$$

The general form of the equation is written as

$$a_C T_4 + a_E T_5 + a_N T_7 + a_S T_1 = b_C$$

where

$$a_E = FluxF_e = -k_e gDiff_e = -3 \times 10^{-3} \times 0.667 = -0.002$$
$$a_N = FluxF_n = -k_n gDiff_n = -10^{-3} \times 1 = -0.001$$
$$a_S = FluxF_s = -k_s gDiff_s = -10^{-3} \times 1 = -0.001$$

The west coefficient does not appear in the equation as its influence is integrated through the boundary condition as

$$FluxC_w = k_w gDiff_w = 10^{-3} \times 2 = 0.002$$
$$FluxV_w = -k_w gDiff_w T_{20} = -10^{-3} \times 2 \times 320 = -0.64$$

The main coefficient and source term can now be calculated and are given by

$$\begin{aligned} a_C &= FLuxC_e + FLuxC_w + FLuxC_n + FLuxC_s \\ &= 0.002 + 0.002 + 0.001 + 0.001 = 0.006 \\ b_C &= -FluxV_w = 0.64 \end{aligned}$$

Substituting, the discretized algebraic equation is obtained as

$$0.006T_4 - 0.002T_5 - 0.001T_7 - 0.001T_1 = 0.64$$

Element #5
The needed diffusion terms are calculated as

$$gDiff_e = \frac{\Delta y_5}{\delta x_{5-6}} = \frac{0.1}{0.25} = 0.4 \quad gDiff_w = \frac{\Delta y_5}{\delta x_{4-5}} = \frac{0.1}{0.15} = 0.667$$
$$gDiff_n = \frac{\Delta x_5}{\delta y_{5-8}} = \frac{0.2}{0.1} = 2 \qquad gDiff_s = \frac{\Delta x_5}{\delta y_{2-5}} = \frac{0.2}{0.1} = 2$$

The interface conductivities are

$$k_e = k_{5-6} = 10^2 \quad k_w = k_{4-5} = 3 \times 10^{-3} \quad k_n = k_{5-8} = 10^2 \quad k_s = k_{2-5} = 10^2$$

The general form of the equation is written as

$$a_C T_5 + a_E T_6 + a_W T_4 + a_N T_8 + a_S T_2 = b_C$$

where

$$a_E = FluxF_e = -k_e gDiff_e = -10^2 \times 0.4 = -40$$
$$a_W = FluxF_w = -k_w gDiff_w = -3 \times 10^{-3} \times 0.667 = -0.002$$
$$a_N = FluxF_n = -k_n gDiff_n = -10^2 \times 2 = -200$$
$$a_S = FluxF_s = -k_s gDiff_s = -10^2 \times 2 = -200$$

All coefficients appear in the equation as this is an internal element. The main coefficient and source term are calculated as

$$\begin{aligned} a_C &= FLuxC_e + FLuxC_w + FLuxC_n + FLuxC_s \\ &= 40 + 0.002 + 200 + 200 = 440.002 \\ b_C &= 0 \end{aligned}$$

Substituting, the discretized algebraic equation is obtained as

$$440.002T_5 - 40T_6 - 0.002T_4 - 200T_8 - 200T_2 = 0$$

Element #6

The needed diffusion terms are calculated as

$$gDiff_w = \frac{\Delta y_6}{\delta x_{5-6}} = \frac{0.1}{0.25} = 0.4 \quad gDiff_n = \frac{\Delta x_6}{\delta y_{6-9}} = \frac{0.3}{0.1} = 3 \quad gDiff_s = \frac{\Delta x_6}{\delta y_{3-6}} = \frac{0.3}{0.1} = 3$$

The interface conductivities are

$$k_w = k_{5-6} = 10^2 \quad k_n = k_{6-9} = 10^2 \quad k_s = k_{3-6} = 10^2$$

The general form of the equation is written as

$$a_C T_6 + a_W T_5 + a_N T_9 + a_S T_3 = b_C$$

where

$$\begin{aligned} a_W &= FluxF_w = -k_w gDiff_w = -10^2 \times 0.4 = -40 \\ a_N &= FluxF_n = -k_n gDiff_n = -10^2 \times 3 = -300 \\ a_S &= FluxF_s = -k_s gDiff_s = -10^2 \times 3 = -300 \end{aligned}$$

The east coefficient does not appear in the equation as its represents a zero flux boundary condition. Moreover the main coefficient and source term are given by

$$\begin{aligned} a_C &= FLuxC_w + FLuxC_n + FLuxC_s \\ &= 40 + 300 + 300 = 640 \\ b_C &= FLuxV_e = 0 \end{aligned}$$

Substituting, the discretized algebraic equation is obtained as

$$640T_6 - 40T_5 - 300T_9 - 300T_3 = 0$$

Element #7
The needed diffusion terms are calculated as

$$gDiff_e = \frac{\Delta y_7}{\delta x_{7-8}} = \frac{0.1}{0.15} = 0.667 \quad gDiff_w = \frac{\Delta y_7}{\delta x_{19-7}} = \frac{0.1}{0.05} = 2 \quad gDiff_s = \frac{\Delta x_7}{\delta y_{4-7}} = \frac{0.1}{0.1} = 1$$

The interface conductivities are

$$k_e = k_{7-8} = 3 \times 10^{-3} \quad k_w = k_{19} = 10^{-3} \quad k_n = k_{18} = 10^{-3} \quad k_s = k_{4-7} = 10^{-3}$$

The general form of the equation is written as

$$a_C T_7 + a_E T_8 + a_S T_4 = b_C$$

where

$$a_E = FluxF_e = -k_e gDiff_e = -3 \times 10^{-3} \times 0.667 = -0.002$$
$$a_S = FluxF_s = -k_s gDiff_s = -10^{-3} \times 1 = -0.001$$

The west and north coefficients do not appear in the equation as their influence is integrated through the boundary conditions as

$$R_{eq} = \frac{h_\infty (k_{18}/\delta y_{7-18})}{h_\infty + (k_{18}/\delta y_{7-18})} \Delta x_7 = \frac{20(10^{-3}/0.05)}{20 + 10^{-3}/0.05} 0.1 = 0.001998$$

$$FluxC_w = k_w gDiff_w = 10^{-3} \times 2 = 0.002 \quad FluxV_w = -k_w gDiff_w T_{19} = -0.64$$
$$FluxC_n = R_{eq} = 0.001998 \quad FluxV_n = -\mathrm{R}_{eq} T_\infty = -0.5994$$

The main coefficient and source term can now be calculated and are given by

$$a_C = FLuxC_e + FLuxC_w + FLuxC_n + FLuxC_s$$
$$= 0.002 + 0.002 + 0.001998 + 0.001 = 0.006998$$
$$b_C = -FLuxV_w - FLuxV_n = 0.64 + 0.5994 = 1.2394$$

Substituting, the discretized algebraic equation is obtained as

$$0.006998 T_7 - 0.002 T_8 - 0.001 T_4 = 1.2394$$

Element #8
The needed diffusion terms are calculated as

$$gDiff_e = \frac{\Delta y_8}{\delta x_{8-9}} = \frac{0.1}{0.25} = 0.4 \quad gDiff_w = \frac{\Delta y_8}{\delta x_{7-8}} = \frac{0.1}{0.15} = 0.667 \quad gDiff_s = \frac{\Delta x_8}{\delta y_{5-8}} = \frac{0.2}{0.1} = 2$$

The interface conductivities are

$$k_e = k_{8-9} = 10^2 \quad k_w = k_{7-8} = 3 \times 10^{-3} \quad k_n = k_{17} = 10^2 \quad k_s = k_{5-8} = 10^2$$

The general form of the equation is written as

$$a_C T_8 + a_E T_9 + a_W T_7 + a_S T_5 = b_C$$

where

$$\begin{aligned} a_E &= FluxF_e = -k_e gDiff_e = -10^2 \times 0.4 = -40 \\ a_W &= FluxF_w = -k_w gDiff_w = -3 \times 10^{-3} \times 0.667 = -0.002 \\ a_S &= FluxF_s = -k_s gDiff_s = -10^2 \times 2 = -200 \end{aligned}$$

The north coefficient does not appear in the equation as its influence is integrated through the boundary condition as

$$R_{eq} = \frac{h_\infty (k_{17}/\delta y_{8-17})}{h_\infty + (k_{17}/\delta y_{8-17})} \Delta x_8 = \frac{20(10^2/0.05)}{20 + 10^2/0.05} 0.2 = 3.9604$$

$$FluxC_n = R_{eq} = 3.9604 \quad FluxV_n = -R_{eq} T_\infty = -1188.12$$

The main coefficient and source term can now be calculated and are given by

$$\begin{aligned} a_C &= FLuxC_e + FLuxC_w + FLuxC_n + FLuxC_s \\ &= 40 + 0.002 + 3.9604 + 200 = 243.9624 \\ b_C &= -FluxV_n = 1188.12 \end{aligned}$$

Substituting, the discretized algebraic equation is obtained as

$$243.9624 T_8 - 40 T_9 - 0.002 T_7 - 200 T_5 = 1188.12$$

Element #9
The needed diffusion terms are calculated as

$$gDiff_w = \frac{\Delta y_9}{\delta x_{8-9}} = \frac{0.1}{0.25} = 0.4 \quad gDiff_s = \frac{\Delta x_9}{\delta y_{6-9}} = \frac{0.3}{0.1} = 3$$

The interface conductivities are

$$k_w = k_{8-9} = 10^2 \quad k_n = k_{16} = 10^2 \quad k_s = k_{6-9} = 10^2$$

The general form of the equation is written as

$$a_C T_9 + a_W T_8 + a_S T_6 = b_C$$

where

$$a_W = FluxF_w = -k_w gDiff_w = -10^2 \times 0.4 = -40$$
$$a_S = FluxF_s = -k_s gDiff_s = -10^2 \times 3 = -300$$

The east and north coefficients do not appear in the equation as their influence is integrated through the boundary conditions as

$$R_{eq} = \frac{h_\infty (k_{16}/\delta y_{9-16})}{h_\infty + (k_{16}/\delta y_{9-16})} \Delta x_9 = \frac{20(10^2/0.05)}{20 + 10^2/0.05} 0.3 = 5.9406$$

$$FluxC_n = R_{eq} = 5.9406$$
$$FluxV_n = -R_{eq} T_\infty = -1782.18$$
$$FluxC_e = FluxV_e = 0$$

The main coefficient and source term can now be calculated and are given by

$$a_C = FLuxC_w + FLuxC_n + FLuxC_s = 40 + 5.9406 + 300 = 345.9406$$
$$b_C = -FLuxV_n = 1782.18$$

Substituting, the discretized algebraic equation is obtained as

$$345.9406T_9 - 40T_8 - 300T_6 = 1782.18$$

Summary of equations
The discretized algebraic equations are given by

$$0.005T_1 - 0.002T_2 - 0.001T_4 = 0.64$$

$$240.002T_2 - 40T_3 - 0.002T_1 - 200T_5 = 20$$

$$340T_3 - 40T_2 - 300T_6 = 30$$

$$0.006T_4 - 0.002T_5 - 0.001T_7 - 0.001T_1 = 0.64$$

$$440.002T_5 - 40T_6 - 0.002T_4 - 200T_8 - 200T_2 = 0$$

$$640T_6 - 40T_5 - 300T_9 - 300T_3 = 0$$

$$0.006998T_7 - 0.002T_8 - 0.001T_4 = 1.2394$$

$$243.9624T_8 - 40T_9 - 0.002T_7 - 200T_5 = 1188.12$$

$$345.9406T_9 - 40T_8 - 300T_6 = 1782.18$$

Solution of Equations

To solve the above system of equations via the Gauss-Seidel method, the equations are rearranged into

$$T_1 = (0.002T_2 + 0.001T_4 + 0.64)/0.005$$

$$T_2 = (40T_3 + 0.002T_1 + 200T_5 + 20)/240.002$$

$$T_3 = (40T_2 + 300T_6 + 30)/340$$

$$T_4 = (0.002T_5 + 0.001T_7 + 0.001T_1 + 0.64)/0.006$$

$$T_5 = (40T_6 + 0.002T_4 + 200T_8 + 200T_2)/440.002$$

$$T_6 = (40T_5 + 300T_9 + 300T_3)/640$$

$$T_7 = (0.002T_8 + 0.001T_4 + 1.2394)/0.006998$$

$$T_8 = (40T_9 + 0.002T_7 + 200T_5 + 1188.12)/243.9624$$

$$T_9 = (40T_8 + 300T_6 + 1782.18)/345.9406$$

The solution is found iteratively with the latest available values used during the solution process. Starting with a uniform initial guess of 300 K, Table 8.1 shows the results during the first two iterations along with the final converged solution.

Table 8.1 Summary of results obtained using the Gauss-Seidel iterative method

Iter	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
0	300	300	300	300	300	300	300	300	300
1	308.000	300.083	300.098	308.000	300.037	300.048	306.859	300.031	300.045
2	309.633	300.131	300.146	309.428	300.078	300.094	307.072	300.071	300.090
⋮	…	…	…	…	…	…	…	…	…
Solution	312.490	305.254	305.254	311.944	305.154	305.154	308.867	305.054	305.054

Values of T along the bottom boundary

These can be calculated from the specified boundary condition, in this case a specified flux, as

$$-k_b \frac{\partial T}{\partial y} = q_b \Rightarrow -k_b \frac{T_C - T_b}{y_C - y_b} = q_b \Rightarrow T_b = T_C + \frac{q_b}{k_b}(y_C - y_b)$$

Using the above equation, the temperatures are calculated as

$$T_{10} = T_1 = 312.490$$
$$T_{11} = T_2 + \frac{q_b}{k_{11}}(y_2 - y_{11}) = 305.254 + \frac{100}{100}(0.05 - 0) = 305.304$$
$$T_{12} = T_3 + \frac{q_b}{k_{12}}(y_3 - y_{12}) = 305.254 + \frac{100}{100}(0.05 - 0) = 305.304$$

Values of T along the right boundary

The above equation with the heat flux set to zero can be used to calculate the temperature along the zero flux boundary, leading to

$$-k_b \frac{\partial T}{\partial y} = 0 \Rightarrow T_b = T_C$$

The boundary temperatures are therefore given by

$$T_{13} = T_3 = 305.254 \quad T_{14} = T_6 = 305.154 \quad T_{15} = T_9 = 305.054$$

Values of T along the top boundary

Using Eq. (8.45) the temperature values along the top boundary are computed as

$$T_{18} = \frac{h_b T_\infty + (k_{18}/\delta y_{7-18}) T_7}{h_b + (k_{18}/\delta y_{7-18})} = \frac{20 \times 300 + (10^{-3}/0.05)308.867}{20 + (10^{-3}/0.05)} = 300.009$$
$$T_{17} = \frac{h_b T_\infty + (k_{17}/\delta y_{8-17}) T_8}{h_b + (k_{17}/\delta y_{8-17})} = \frac{20 \times 300 + (10^{2}/0.05)305.054}{20 + (10^{2}/0.05)} = 305.004$$
$$T_{16} = \frac{h_b T_\infty + (k_{16}/\delta y_{9-16}) T_9}{h_b + (k_{16}/\delta y_{9-16})} = \frac{20 \times 300 + (10^{2}/0.05)305.054}{20 + (10^{2}/0.05)} = 305.004$$

Total heat transfer along the left boundary
The total heat transfer along the west boundary is given by

$$\begin{aligned} Q_{Left} &= q_{21}\Delta y_{21} + q_{20}\Delta y_{20} + q_{19}\Delta y_{19} \\ &= k_{21}\frac{T_1 - T_{21}}{\delta x_{21-1}}\Delta y_{21} + k_{20}\frac{T_4 - T_{20}}{\delta x_{20-4}}\Delta y_{20} + k_{19}\frac{T_7 - T_{19}}{\delta x_{19-7}}\Delta y_{19} \\ &= \frac{10^{-3}\times 0.1}{0.05}[312.49 + 311.944 + 308.867 - 3\times 320] \\ &= -0.053398\,\mathrm{W} \end{aligned}$$

Total heat transfer along the top boundary
Along the top boundary, the total heat transfer is computed as

$$\begin{aligned} Q_{top} &= q_{18}\Delta x_{18} + q_{17}\Delta x_{17} + q_{16}\Delta x_{16} \\ &= h_\infty[\Delta x_{18}(T_{18} - T_\infty) + \Delta x_{17}(T_{17} - T_\infty) + \Delta x_{16}(T_{16} - T_\infty)] \\ &= 20[0.1(300.009 - 300) + 0.2(305.004 - 300) + 0.3(305.004 - 300)] \\ &= 50.058\,\mathrm{W} \end{aligned}$$

Total heat transfer along the bottom boundary
Knowing the heat flux, the total amount of heat transfer is found as

$$Q_{Bottom} = -q_b\Delta y = -100\times 0.5 = -50\,\mathrm{W}$$

Check for energy conservation
For conservation, the sum of heat entering and leaving the domain should be equal to zero. The sum is computed as

$$Q_{Top} + Q_{Left} + Q_{Bottom} = 50.058 - 0.053398 - 50 = 0.004602 \approx 0$$

The slight deviation from zero is due to the use of a limited number of digits during computations.

8.5 Non-Cartesian Orthogonal Grids

Let us now consider the case of an orthogonal grid that is not oriented along the x and y axes. As shown in Fig. 8.8, such a grid can be obtained by rotating the Cartesian grid of Fig. 8.1 by some angle.

The discretized equation for this grid should be exactly the same as the one obtained for the Cartesian grid. Moreover for similar boundary conditions, the same solution should be obtained.

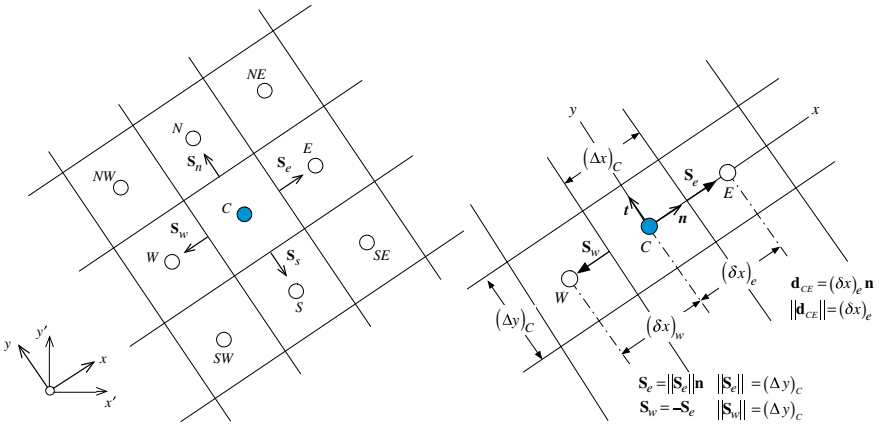

Fig. 8.8 Example of non-Cartesian orthogonal grids

Consider again the steady state conduction Eq. (8.1)

$$\nabla \cdot \mathbf{J}^{\phi,D} = Q^{\phi} \tag{8.58}$$

As before its discretized form is given by

$$\sum_{f \sim nb(C)} -\left(\Gamma^{\phi} \nabla \phi\right)_f \cdot \mathbf{S}_f = Q_C^{\phi} V_C \tag{8.59}$$

Considering the discretization along face e, the following is obtained:

$$\mathbf{J}_e^{\phi,D} \cdot \mathbf{S}_e = -\Gamma_e^{\phi} (\nabla \phi \cdot \mathbf{n})_e S_e = -\Gamma_e^{\phi} \left(\frac{\partial \phi}{\partial n}\right)_e S_e \tag{8.60}$$

where now

$$(\nabla \phi \cdot \mathbf{n})_e = \left(\frac{\partial \phi}{\partial n}\right)_e \tag{8.61}$$

is the gradient of ϕ at face e along the $\mathbf{n}$ direction. Assuming again a linear profile for ϕ along the $\mathbf{n}$ coordinate axis, the gradient can be written as

$$\left(\frac{\partial \phi}{\partial n}\right)_e = \frac{\phi_E - \phi_C}{d_{CE}} \tag{8.62}$$

The discretization of other terms proceeds as for a Cartesian grid, leading to the same final discretization equation.

8.6 Non-orthogonal Unstructured Grid

8.6.1 Non-orthogonality

In the above configurations, the fluxes were normal to the face. In general, structured curvilinear grids and unstructured grids are non-orthogonal [2–5].Therefore the surface vector $\mathbf{S}_f$ and the vector **CF** joining the centroids of the elements straddling the interface are not collinear (see Fig. 8.9). In this case the gradient normal to the surface cannot be written as a function of ϕ_F and ϕ_C, as it has a component in the direction perpendicular to **CF**.

Thus while on orthogonal grids the gradient in the direction normal to the interface yields

$$(\nabla\phi \cdot \mathbf{n})_f = \left(\frac{\partial\phi}{\partial n}\right)_f = \frac{\phi_F - \phi_C}{\|\mathbf{r}_F - \mathbf{r}_C\|} = \frac{\phi_F - \phi_C}{d_{CF}} \tag{8.63}$$

because **CF** and **n** (the unit vector normal to the surface) are aligned, on non-orthogonal grids [6, 7], the gradient direction that yields an expression involving ϕ_F and ϕ_C will have to be along the line joining the two points C and F.

If **e** represents the unit vector along the direction defined by the line connecting nodes C and F then

$$\mathbf{e} = \frac{\mathbf{r}_F - \mathbf{r}_C}{\|\mathbf{r}_F - \mathbf{r}_C\|} = \frac{\mathbf{d}_{CF}}{d_{CF}} \tag{8.64}$$

Therefore, the gradient in the **e** direction can be written as

$$(\nabla\phi \cdot \mathbf{e})_f = \left(\frac{\partial\phi}{\partial e}\right)_f = \frac{\phi_F - \phi_C}{\|\mathbf{r}_\mathbf{F} - \mathbf{r}_C\|} = \frac{\phi_F - \phi_C}{d_{CF}} \tag{8.65}$$

Thus to achieve the linearization of the flux in non-orthogonal grids, the surface vector $\mathbf{S}_f$ should be written as the sum of two vectors $\mathbf{E}_f$ and $\mathbf{T}_f$, i.e.,

$$\mathbf{S}_f = \mathbf{E}_f + \mathbf{T}_f \tag{8.66}$$

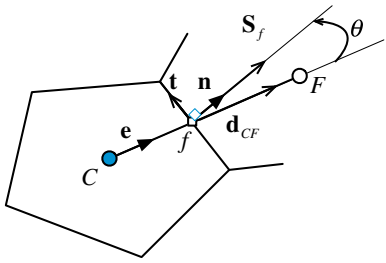

Fig. 8.9 An element in a non-orthogonal mesh system

with $\mathbf{E}_f$ being in the **CF** direction to enable writing part of the diffusion flux as a function of the nodal values ϕ_F and ϕ_C such that

$$\begin{aligned}(\nabla\phi)_f \cdot \mathbf{S}_f &= \underbrace{(\nabla\phi)_f \cdot \overbrace{\mathbf{E}_f}^{E_f\mathbf{e}}}_{orthogonal-like\ contribution} + \underbrace{(\nabla\phi)_f \cdot \mathbf{T}}_{non-orthogonal\ like\ contribution} \\ &= E_f \left(\frac{\partial\phi}{\partial e}\right)_f + (\nabla\phi)_f \cdot \mathbf{T}_f \\ &= E_f \frac{\phi_F - \phi_C}{d_{CF}} + (\nabla\phi)_f \cdot \mathbf{T}_f\end{aligned} \tag{8.67}$$

The first term on the right hand side of Eq. (8.67) represents a contribution similar to the contribution on orthogonal grids, i.e., involving ϕ_F and ϕ_C, while the second term on the right hand side is called cross-diffusion or non-orthogonal diffusion [8] and is due to the non-orthogonality of the grid. Different options for the decomposition of $\mathbf{S}_f$ are available and are discussed next.

8.6.2 Minimum Correction Approach

As shown in Fig. 8.10, the decomposition of $\mathbf{S}_f$ is done in such a way as to keep the non-orthogonal correction in Eq. (8.67) as small as possible, by making $\mathbf{E}_f$ and $\mathbf{T}_f$ orthogonal. As the non-orthogonality increases, the contribution to the diffusion flux from ϕ_F and ϕ_C decreases. In this case the vector $\mathbf{E}_f$ is computed as

$$\mathbf{E}_f = (\mathbf{e} \cdot \mathbf{S}_f)\mathbf{e} = (S_f \cos\theta)\mathbf{e} \tag{8.68}$$

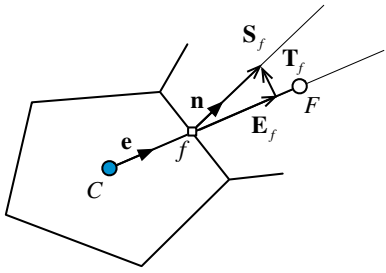

Fig. 8.10 Decomposing $\mathbf{S}_f$ via the minimum correction approach

8.6.3 Orthogonal Correction Approach

This approach, schematically depicted in Fig. 8.11, keeps the contribution of the term involving ϕ_F and ϕ_C the same as on an orthogonal mesh irrespective of the degree of grid non-orthogonality. To achieve this, $\mathbf{E}_f$ is defined as

$$\mathbf{E}_f = S_f \mathbf{e} \tag{8.69}$$

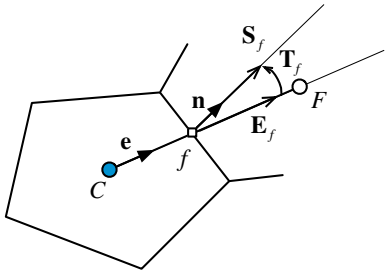

Fig. 8.11 Decomposing $\mathbf{S}_f$ via the orthogonal correction approach

8.6.4 Over-Relaxed Approach

In this approach, the importance of the term involving ϕ_F and ϕ_C is forced to increase as grid non-orthogonality increases. As shown in Fig. 8.12, this is achieved by selecting $\mathbf{T}_f$ to be normal to $\mathbf{S}_f$. Mathematically $\mathbf{E}_f$ is computed as

$$\mathbf{E}_f = \left(\frac{S_f}{\cos\theta}\right)\mathbf{e} = \left(\frac{S_f^2}{S_f \cos\theta}\right)\mathbf{e} = \frac{\mathbf{S}_f \cdot \mathbf{S}_f}{\mathbf{e} \cdot \mathbf{S}_f}\mathbf{e} \tag{8.70}$$

To summarize, the diffusion flux at an element face of a non-orthogonal grid cannot be written solely in terms of the values at the nodes straddling the face.

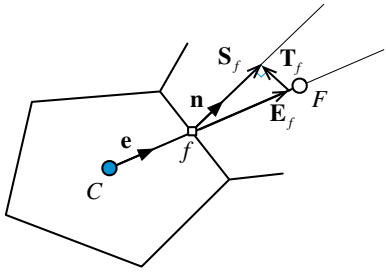

Fig. 8.12 Decomposing $\mathbf{S}_f$ via the over-relaxed approach

A term that accounts for non-orthogonality has to be added. This term is denoted in the literature by "cross diffusion" and is computed as

$$(\nabla\phi)_f \cdot \mathbf{T}_f = (\nabla\phi)_f \cdot \left(\mathbf{S}_f - \mathbf{E}_f\right) = \begin{cases} (\nabla\phi)_f \cdot (\mathbf{n} - \cos\theta\mathbf{e})S_f & \textit{minimum correction} \\ (\nabla\phi)_f \cdot (\mathbf{n} - \mathbf{e})S_f & \textit{normal correction} \\ (\nabla\phi)_f \cdot \left(\mathbf{n} - \dfrac{1}{\cos\theta}\mathbf{e}\right)S_f & \textit{over-relaxed} \end{cases} \tag{8.71}$$

For orthogonal meshes $\mathbf{n}$ and $\mathbf{e}$ are collinear, the angle θ shown in Fig. 8.9 is zero, and the cross-diffusion term is zero. When cross diffusion is not zero, and since it cannot be written as a function of ϕ_F and ϕ_C, it is added as a source term in the element algebraic equation.

All approaches described above are correct and satisfy Eq. (8.59). The difference between these methods is in their accuracy and stability on non-orthogonal meshes. The over-relaxed approach has been found to be the most stable even when the grid is highly non-orthogonal. Nevertheless, the final general form of the discretized diffusion term is the same for all three approximations.

8.6.5 Treatment of the Cross-Diffusion Term

The cross diffusion term cannot be expressed in terms of nodal values. Due to this fact, it is treated in a deferred correction manner by computing its value using the current gradient field and adding it as a source term on the right hand side of the algebraic equation. Gradients are computed at the main grid points, as described next, and values at the interfaces are obtained by interpolation.

8.6.6 Gradient Computation

In discretizing the diffusion term over a one-dimensional or orthogonal multi-dimensional computational domain, it was shown that the gradient of ϕ can be explicitly written as a function of the cell nodal ϕ values. In non-orthogonal domains however, the computation of the diffusion flux was found to be more complex. The non-orthogonal component of the gradient could not be linearized and written as a function of nodal values, but rather had to be moved to the right hand side and evaluated explicitly. This means that the gradient has to be evaluated in order to incorporate its non-orthogonal contribution in the discretization equation. One widely used approach for computing the gradient at a cell is the

Green-Gauss or gradient theorem, which states that for any closed volume V, surrounded by a surface ∂V the following holds:

$$\int_V \nabla\phi \, dV = \oint_{\partial V} \phi \, d\mathbf{S} \tag{8.72}$$

where $d\mathbf{S}$ is the outward pointing incremental surface vector. In order to obtain a discrete version of this equation, the mean value theorem is applied according to which the integral on the left hand side of Eq. (8.72) and the average gradient over the volume $\boldsymbol{V}$ are related by

$$\overline{\nabla\phi} V = \int_V \nabla\phi \, dV \tag{8.73}$$

Combining Eqs. (8.72) and (8.73), the average gradient over element C shown in Fig. 8.9 is found to be

$$\overline{\nabla\phi}_C = \frac{1}{V_C} \oint_{\partial V_C} \phi_f \mathbf{S}_f \tag{8.74}$$

Next the integral over a cell face is approximated by the face centroid value times the face area. Thus $\overline{\nabla\phi}_C$, or simply $\nabla\phi_C$, is computed as

$$\nabla\phi_C = \frac{1}{V_C} \sum_{f \sim nb(C)} \phi_f \mathbf{S}_f \tag{8.75}$$

The gradient at the face of an element can then be obtained as the weighted average of the gradients at the centroids straddling the interface and is given by

$$\nabla\phi_f = g_C \nabla\phi_C + g_F \nabla\phi_F \tag{8.76}$$

where g_C and g_F are geometric interpolation factors related to the position of the element face f with respect to the nodes C and F.

8.6.7 Algebraic Equation for Non-orthogonal Meshes

Splitting the surface vector $\mathbf{S}_f$ into the two $\mathbf{E}_f$ and $\mathbf{T}_f$ vectors and substituting its equivalent expression into the semi-discretized equation of the diffusion fluxes, yield

$$\begin{aligned}\sum_{f\sim nb(C)}\left(\mathbf{J}_f^{\phi,D}\cdot\mathbf{S}_f\right) &= \sum_{f\sim nb(C)}\left(-\left(\Gamma^{\phi}\nabla\phi\right)_f\cdot\left(\mathbf{E}_f+\mathbf{T}_f\right)\right)\\ &= \underbrace{\sum_{f\sim nb(C)}\left(-\left(\Gamma^{\phi}\nabla\phi\right)_f\cdot\mathbf{E}_f\right)}_{Orthogonal\ Linearizeable\ Part}+\underbrace{\sum_{f\sim nb(C)}\left(-\left(\Gamma^{\phi}\nabla^{\phi}\right)_f\cdot\mathbf{T}_f\right)}_{Non-Orthogonal\ Non-Linearizeable\ Part}\\ &= \sum_{f\sim nb(C)}\left(-\Gamma_f^{\phi}E_f\frac{(\phi_F-\phi_C)}{d_{CF}}\right)+\sum_{f\sim nb(C)}\left(-\left(\Gamma^{\phi}\nabla\phi\right)_f\cdot\mathbf{T}_f\right)\\ &= \sum_{f\sim nb(C)}\Gamma_f^{\phi}gDiff_f(\phi_C-\phi_F)+\sum_{f\sim nb(C)}\left(-\left(\Gamma^{\phi}\nabla\phi\right)_f\cdot\mathbf{T}_f\right)\\ &= \left(\sum_{f\sim nb(C)}FluxC_f\right)\phi_C+\sum_{f\sim nb(C)}\left(FluxF_f\phi_F\right)+\sum_{f\sim nb(C)}\left(FluxV_f\right)\end{aligned} \tag{8.77}$$

where again $gDiff_f$ is a geometric diffusion coefficient defined as

$$gDiff_f=\frac{E_f}{d_{CF}} \tag{8.78}$$

Using the above form of the diffusion fluxes and expanding, the final form of the discretized diffusion equation over unstructured/structured non-orthogonal grid is obtained as

$$a_C\phi_C+\sum_{F\sim NB(C)}a_F\phi_F=b_C \tag{8.79}$$

where

$$\begin{aligned}a_F &= FluxF_f=-\Gamma_f^{\phi}gDiff_f\\ a_C &= \sum_{f\sim nb(C)}FluxC_f=-\sum_{f\sim nb(C)}FluxF_f=\sum_{f\sim nb(C)}\Gamma_f^{\phi}gDiff_f\\ b_C &= Q_C^{\phi}V_C-\sum_{f\sim nb(C)}\left(FluxV_f\right)=Q_C^{\phi}V_C+\sum_{f\sim nb(C)}\left(\left(\Gamma^{\phi}\nabla\phi\right)_f\cdot\mathbf{T}_f\right)\end{aligned} \tag{8.80}$$

Note the change in sign of the non-orthogonal term on the right hand side of the equation.

Example 2

For the polygonal element C and its neighbor F shown in Fig. 8.13, the solution at any point satisfies $\phi = x^2 + y^2 + x^2y^2$. *If the volume of cell C is* $V_C = 8.625$ *calculate the following:*

1. *The gradient of* ϕ *at* (i.e., $\nabla\phi_C$) *both numerically and analytically.*
2. *The analytical value of* $\nabla\phi_F$.
3. *Interpolate between the numerical value of* $\nabla\phi_C$ *and the analytical value of* $\nabla\phi_F$ *to find an approximate value for* $\nabla\phi_{f_1}$. *Compare with the analytical value of* $\nabla\phi_{f_1}$.
4. *Express* $\nabla\phi_{f_1} \cdot \mathbf{S}_{f_1}$ *in terms of* ϕ_C *and* ϕ_F *using*
   (a) *The minimum correction approach*
   (b) *The orthogonal correction approach*
   (c) *The over-relaxed approach*

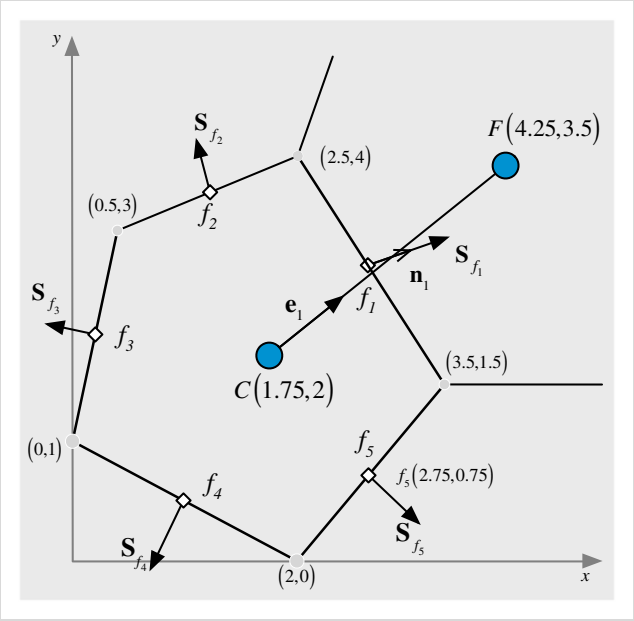

Fig. 8.13 A polygonal element with its geometric entities

Solution

1. Knowing that $\phi = x^2 + y^2 + x^2y^2$ at any point in the domain, the value of the gradient at any point can be calculated as

$$\nabla\phi = \frac{\partial\phi}{\partial x}\mathbf{i} + \frac{\partial\phi}{\partial y}\mathbf{j} = \left(2x + 2xy^2\right)\mathbf{i} + \left(2y + 2yx^2\right)\mathbf{j}$$

Therefore the analytical value of $\nabla\phi_C$ is given by

$$\nabla\phi_C = 17.5\mathbf{i} + 16.25\mathbf{j}$$

The numerical value of $\nabla\phi_C$ can be computed using

$$\nabla\phi_C = \frac{1}{V_C}\sum_{f\sim nb(C)} \phi_f \mathbf{S}_f$$

Therefore the values of ϕ_f and $\mathbf{S}_f$ are required. Using the given analytical expression, the values of ϕ at the various locations are found as

$$\phi_C = 1.75^2 + 2^2 + 1.75^2 \times 2^2 = 19.3125$$

$$\phi_F = 4.25^2 + 3.5^2 + 4.25^2 \times 3.5^2 = 251.578125$$

$$\phi_{f_1} = 3^2 + 2.75^2 + 3^2 \times 2.75^2 = 84.625$$

$$\phi_{f_2} = 1.5^2 + 3.5^2 + 1.5^2 \times 3.5^2 = 42.0625$$

$$\phi_{f_3} = 0.25^2 + 2^2 + 0.25^2 \times 2^2 = 4.3125$$

$$\phi_{f_4} = 1^2 + 0.5^2 + 1^2 \times 0.5^2 = 1.5$$

$$\phi_{f_5} = 2.75^2 + 0.75^2 + 2.75^2 \times 0.75^2 = 12.37890625$$

The surface vector is given by

$$\mathrm{S}_f = \pm(\Delta y\mathbf{i} - \Delta x\mathbf{j})$$

where the correct sign is chosen such that the vector is pointing outward. This is done by computing the surface vector as $\mathbf{S}_f = \Delta y\mathbf{i} - \Delta x\mathbf{j}$ and then performing the dot product of $\mathbf{S}_f$ with the distance vector $\mathbf{d}_{CF}$ (pointing from C to F). If the product is positive then the direction of $\mathbf{S}_f$ is correct. If the product is negative then the sign of $\mathbf{S}_f$ should be reversed, as shown below.

The distance vectors are computed as

$$\mathbf{d}_{Cf_1} = (3 - 1.75)\mathbf{i} + (2.75 - 2)\mathbf{j} = 1.25\mathbf{i} + 0.75\mathbf{j}$$

$$\mathbf{d}_{Cf_2} = (1.5 - 1.75)\mathbf{i} + (3.5 - 2)\mathbf{j} = -0.25\mathbf{i} + 1.5\mathbf{j}$$

$$\mathbf{d}_{Cf_3} = (0.25 - 1.75)\mathbf{i} + (2 - 2)\mathbf{j} = -1.5\mathbf{i}$$

$$\mathbf{d}_{Cf_4} = (1 - 1.75)\mathbf{i} + (0.5 - 2)\mathbf{j} = -0.75\mathbf{i} - 1.5\mathbf{j}$$

$$\mathbf{d}_{Cf_5} = (2.75 - 1.75)\mathbf{i} + (0.75 - 2)\mathbf{j} = \mathbf{i} - 1.25\mathbf{j}$$

while the tentative surface vectors are given by

$$\mathbf{S}_{f_1} = (4 - 1.5)\mathbf{i} - (2.5 - 3.5)\mathbf{j} = 2.5\mathbf{i} + \mathbf{j}$$

$$\mathbf{S}_{f_2} = (4 - 3)\mathbf{i} - (2.5 - 0.5)\mathbf{j} = \mathbf{i} - 2\mathbf{j}$$

$$\mathbf{S}_{f_3} = (3 - 1)\mathbf{i} - (0.5 - 0)\mathbf{j} = 2\mathbf{i} - 0.5\mathbf{j}$$

$$\mathbf{S}_{f_4} = (1 - 0)\mathbf{i} - (0 - 2)\mathbf{j} = \mathbf{i} + 2\mathbf{j}$$

$$\mathbf{S}_{f_5} = (1.5 - 0)\mathbf{i} - (3.5 - 2)\mathbf{j} = 1.5\mathbf{i} - 1.5\mathbf{j}$$

Performing the dot products, the obtained values are

$$\mathbf{S}_{f_1} \cdot \mathbf{d}_{Cf_1} = (2.5\mathbf{i} + \mathbf{j}) \cdot (1.25\mathbf{i} + 0.75\mathbf{j}) = 3.875 > 0$$

$$\mathbf{S}_{f_2} \cdot \mathbf{d}_{Cf_2} = (\mathbf{i} - 2\mathbf{j}) \cdot (-0.25\mathbf{i} + 1.5\mathbf{j}) = -3.25 < 0$$

$$\mathbf{S}_{f_3} \cdot \mathbf{d}_{Cf_3} = (2\mathbf{i} - 0.5\mathbf{j}) \cdot (-1.5\mathbf{i}) = -3 < 0$$

$$\mathbf{S}_{f_4} \cdot \mathbf{d}_{Cf_4} = (\mathbf{i} + 2\mathbf{j}) \cdot (-0.75\mathbf{i} - 1.5\mathbf{j}) = -3.75 < 0$$

$$\mathbf{S}_{f_5} \cdot \mathbf{d}_{Cf_5} = (1.5\mathbf{i} - 1.5\mathbf{j}) \cdot (\mathbf{i} - 1.25\mathbf{j}) = 3.375 > 0$$

Therefore the correct values of the surface vectors should be

$$\mathbf{S}_{f_1} = 2.5\mathbf{i} + \mathbf{j} \quad \mathbf{S}_{f_2} = -\mathbf{i} + 2\mathbf{j} \quad \mathbf{S}_{f_3} = -2\mathbf{i} + 0.5\mathbf{j} \quad \mathbf{S}_{f_4} = -\mathbf{i} - 2\mathbf{j} \quad \mathbf{S}_{f_5} = 1.5\mathbf{i} - 1.5\mathbf{j}$$

Thus, the numerical value of the gradient at C is obtained as

$$\begin{aligned}\nabla\phi_C &= \frac{1}{8.625}\begin{bmatrix} 84.625(2.5\mathbf{i} + \mathbf{j}) + 42.0625(-\mathbf{i} + 2\mathbf{j}) + 4.3125(-2\mathbf{i} + 0.5\mathbf{j}) \\ +1.5(-\mathbf{i} - 2\mathbf{j}) + 12.37890625(1.5\mathbf{i} - 1.5\mathbf{j}) \end{bmatrix} \\ &= 20.63111\mathbf{i} + 17.31454\mathbf{j}\end{aligned}$$

which is close to the analytical value.

2. The analytical value of $\nabla\phi_F$ is easily calculated as

$$\nabla\phi_F = 112.625\mathbf{i} + 133.4375\mathbf{j}$$

3. The interpolated value of $\nabla\phi_{f_1}$ is obtained using

$$\nabla\phi_{f_1} = g_{f_1}\nabla\phi_F + (1 - g_{f_1})\nabla\phi_C$$

where

$$g_{f_1} = \frac{d_{Cf_1}}{d_{Cf_1} + d_{f_1F}}$$

Performing the calculations, the interpolation factor is found to be given by

$$d_{Cf_1} = \sqrt{(x_{f_1} - x_C)^2 + (y_{f_1} - y_C)^2} = \sqrt{(3 - 1.75)^2 + (2.75 - 2)^2} = 1.4577$$
$$d_{fF} = \sqrt{(x_F - x_{f_1})^2 + (y_F - y_{f_1})^2} = \sqrt{(4.25 - 3)^2 + (3.5 - 2.75)^2} = 1.4577$$
$$g_{f_1} = \frac{1.4577}{1.4577 + 1.4577} = 0.5$$

leading to the following value for $\nabla\phi_{f_1}$:

$$\nabla\phi_{f_1} = 66.628055\mathbf{i} + 75.37602\mathbf{j}$$

The analytical value of $\nabla\phi_{f_1}$ is easily obtained as

$$\nabla\phi_{f_1} = 51.375\mathbf{i} + 55\mathbf{j}$$

Again values are close.

4. The general form of $\nabla\phi_{f_1} \cdot \mathbf{S}_{f_1}$ is given by

$$-\nabla\phi_{f_1} \cdot \mathbf{S}_{f_1} = \underbrace{E_{f_1}\frac{(\phi_C - \phi_F)}{d_{CF}}}_{\substack{orthogonal-like \\ contribution}} + \underbrace{-\nabla\phi_{f_1} \cdot \mathbf{T}_{f_1}}_{\substack{non-orthogonal\ like \\ contribution}}$$

with

$$\mathbf{d}_{CF} = (4.25 - 1.75)\mathbf{i} + (3.25 - 2)\mathbf{j} = 2.5\mathbf{i} + 1.5\mathbf{j} \Rightarrow d_{CF} = \sqrt{2.5^2 + 1.5^2} = 2.9155$$

The unit vector $\mathbf{e}_1$ in the direction of $\mathbf{d}_{CF}$ is calculated using

$$\mathbf{e}_1 = \frac{\mathbf{d}_{CF}}{d_{CF}} = 0.8575\mathbf{i} + 0.5145\mathbf{j}$$

(a) The minimum correction approach
Using this approach, the expression for $\mathbf{E}_{f_1}$ is computed as

$$\mathbf{E}_{f_1} = \left(\mathbf{e}_1 \cdot \mathbf{S}_{f_1}\right)\mathbf{e}_1 = 2.279\mathbf{i} + 1.368\mathbf{j} \Rightarrow E_{f_1} = 2.658$$

The normal component is computed as

$$\mathbf{T}_{f_1} = \mathbf{S}_{f_1} - \mathbf{E}_{f_1} = 0.221\mathbf{i} - 0.368\mathbf{j} \Rightarrow \nabla\phi_{f_1} \cdot \mathbf{T}_{f_1} = -13.014$$

The diffusion flux at the face becomes

$$-\nabla\phi_{f_1} \cdot \mathbf{S}_{f_1} = 0.9122(\phi_C - \phi_F) + 13.014$$

(b) The orthogonal correction approach
In this approach, the expression for $\mathbf{E}_{f_1}$ is computed as

$$\mathbf{E}_{f_1} = S_{f_1}\mathbf{e}_1 = 2.309\mathbf{i} + 1.385\mathbf{j} \Rightarrow E_{f_1} = 2.693$$

The normal component is found to be

$$\mathbf{T}_{f_1} = 0.191\mathbf{i} - 0.385\mathbf{j} \Rightarrow \nabla\phi_{f_1} \cdot \mathbf{T}_{f_1} = -16.294$$

The diffusion flux at the face becomes

$$-\nabla\phi_{f_1} \cdot \mathbf{S}_{f_1} = 0.924(\phi_C - \phi_F) + 16.294$$

(c) The over-relaxed approach
The expression for $\mathbf{E}_{f_1}$ is computed as

$$\mathbf{E}_{f_1} = \frac{\mathbf{S}_{f_1} \cdot \mathbf{S}_{f_1}}{\mathbf{e}_1 \cdot \mathbf{S}_{f_1}}\mathbf{e}_1 = 2.339\mathbf{i} + 1.403\mathbf{j} \Rightarrow E_{f_1} = 2.728$$

The normal component is found to be

$$\mathbf{T}_{f_1} = 0.161\mathbf{i} - 0.403\mathbf{j} \Rightarrow \nabla\phi_{f_1} \cdot \mathbf{T}_{f_1} = -19.649$$

The diffusion flux at the face becomes

$$\begin{aligned} -\nabla\phi_{f_1} \cdot \mathbf{S}_{f_1} &= FluxC_{f_1}\phi_C + FluxF_{f_1}\phi_F + FluxV_{f_1} \\ &= 0.936(\phi_C - \phi_F) + 19.649 \end{aligned}$$

8.6.8 Boundary Conditions for Non-orthogonal Grids

The treatment of boundary conditions for non-orthogonal grids is similar to that for orthogonal grids with some minor differences related to the non-orthogonal diffusion contribution. This is outlined next.

8.6.8.1 Dirichlet Boundary Condition

For the case of a Dirichlet boundary condition, i.e., when ϕ is specified by the user at the boundary as shown in Fig. 8.14, the boundary discretization proceeds as in orthogonal-grids. However, there is a need now to account for the cross-diffusion, which arises on boundary faces as on interior faces. This happens whenever the surface vector is not collinear with the vector joining the centroids of the element and boundary face. The diffusion flux along the boundary face is discretized as

$$\begin{aligned}\mathbf{J}_b^{\phi,D}\cdot\mathbf{S}_b &= -\Gamma_b^{\phi}(\nabla\phi)_b\cdot\mathbf{S}_b = -\Gamma_b^{\phi}(\nabla\phi)_b\cdot(\mathbf{E}_b+\mathbf{T}_b)\\ &= -\Gamma_b^{\phi}\left(\frac{\phi_b-\phi_C}{d_{Cb}}\right)E_b-\Gamma_b^{\phi}(\nabla\phi)_b\cdot\mathbf{T}_b\\ &= FluxC_b\phi_C+FluxV_b\end{aligned} \tag{8.81}$$

where

$$\begin{aligned}FluxC_b &= \Gamma_b^{\phi}gDiff_b\\ FluxV_b &= -\Gamma_b^{\phi}gDiff_b\phi_b-\Gamma_b^{\phi}(\nabla\phi)_b\cdot\mathbf{T}_b\\ gDiff_b &= \frac{E_b}{d_{Cb}}\end{aligned} \tag{8.82}$$

Substituting into Eq. (8.79), the modified coefficients are obtained as

$$\begin{aligned}a_F &= FluxF_f \quad a_C = \sum_{f\sim nb(C)} FluxC_f+FluxC_b\\ b_C &= Q_C^{\phi}V_C-FluxV_b-\sum_{f\sim nb(C)} FluxV_f\end{aligned} \tag{8.83}$$

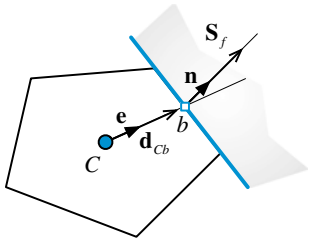

Fig. 8.14 Dirichlet boundary for a non-orthogonal mesh

8.6.8.2 Neumann Boundary Condition

The Neumann type condition for non-orthogonal grids follows that for orthogonal grids. In this case the user-specified flux at the boundary is just added as a source term as with orthogonal grid. The algebraic equation at the boundary is given by Eq. (8.40) with the modified coefficients specified by Eq. (8.41).

8.6.8.3 Mixed Boundary Condition

For the mixed boundary condition case (Fig. 8.5), denoting the convection transfer coefficient by h_∞ and the surrounding value of ϕ by ϕ_∞, the diffusion flux at the boundary can be written as

$$\begin{aligned}\mathbf{J}_b^{\phi,D}\cdot\mathbf{S}_b &= -\Gamma_b^\phi\left(\frac{\phi_b-\phi_C}{d_{Cb}}\right)E_b-\Gamma_b^\phi(\nabla\phi)_b\cdot\mathbf{T}_b\\ &= -h_\infty(\phi_\infty-\phi_b)S_b\end{aligned} \tag{8.84}$$

from which an equation for ϕ_b is obtained as

$$\phi_b=\frac{h_\infty S_b\phi_\infty+\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}\phi_C-\Gamma_b^\phi(\nabla\phi)_b\cdot\mathbf{T}_b}{h_\infty S_b+\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}} \tag{8.85}$$

Substituting ϕ_b back in Eq. (8.84), the flux equation is transformed to

$$\begin{aligned}\mathbf{J}_b^{\phi,D}\cdot\mathbf{S}_b &= \underbrace{-\left[\frac{h_\infty S_b\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}}{h_\infty S_b+\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}}\right]}_{a_b}(\phi_\infty-\phi_C)\underbrace{-\frac{h_\infty S_b\Gamma_b^\phi(\nabla\phi)_b\cdot\mathbf{T}_b}{h_\infty S_b+\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}}}_{S_b^{\phi,CD}}\\ &= FluxC_b\phi_C+FluxV_b\end{aligned} \tag{8.86}$$

where now

$$\begin{aligned}FluxC_b &= \frac{h_\infty S_b\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}}{h_\infty S_b+\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}}\\ FluxV_b &= -FluxC_b\phi_\infty-\frac{h_\infty S_b\Gamma_b^\phi(\nabla\phi)_b\cdot\mathbf{T}_b}{h_\infty S_b+\dfrac{\Gamma_b^\phi E_b}{d_{Cb}}}\end{aligned} \tag{8.87}$$

and the modified coefficients for the boundary element are obtained as

$$\begin{aligned} a_F &= FluxF_f - \Gamma_f^{\phi} \frac{E_f}{d_{Cf}} \\ a_C &= FluxC_b + \sum_{f \sim nb(C)} FluxC_f \\ b_C &= Q_C^{\phi} V_C - FluxV_b - \sum_{f \sim nb(C)} FluxV_f \end{aligned} \tag{8.88}$$

8.7 Skewness

To evaluate many of the terms constituting the general discretized equation of a quantity ϕ, it is necessary to estimate its value at element faces. An estimated value at the face should be the average value of the entire face. At different steps of the discretization process, linear variation of variables between nodes is assumed. If this is extended to variation of variables along the face, then the average value of any variable ϕ should be found at the face centroid. The common practice is to use a linear interpolation profile and to estimate the value at the face at the intersection between the face and the line connecting the two nodes straddling the face. When the grid is skewed the line does not necessarily pass through the centroid of the face [9, 10]. An example is depicted in Fig. 8.15 where the intersection point of segment [*CF*] with the face is at point, *f*′, which does not coincide with the face centroid, *f*. To keep the overall accuracy of the discretization method second order, all face integrations need to take place at point *f*. Thus a correction for the interpolated value at *f*′ is needed in order to get the value at *f*.

The skewness correction is derived by expressing the value of ϕ at *f* in terms of its value and the value of its derivative at *f*′ via a Taylor expansion such that

$$\phi_f = \phi_{f'} + (\nabla\phi)_{f'} \cdot \mathbf{d}_{f'f} \tag{8.89}$$

where $\mathbf{d}_{f'f}$ is a vector from the intersection point *f*′ to the face centre *f*.

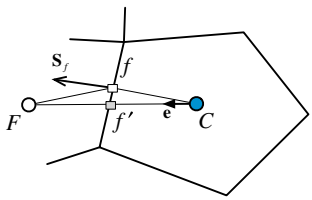

Fig. 8.15 Non-conjunctional elements

8.8 Anisotropic Diffusion

The diffusion equation presented so far assumed the material has no preferred direction for transfer of ϕ with the same diffusion coefficient in all directions, i.e., the medium was assumed to be isotropic. For the case when the diffusion coefficient of the medium is direction dependent, diffusion is said to be anisotropic [11–16]. As mentioned in Chap. 3, some solids are anisotropic for which the semi-discretized diffusion equation becomes

$$\sum_{f\sim nb(C)} \left(-\boldsymbol{\kappa}^{\phi}\cdot\nabla\phi\right)_f \cdot \mathbf{S}_f = S_C^{\phi} V_C \tag{8.90}$$

where $\boldsymbol{\kappa}^{\phi}$ is a second order symmetric tensor. Assuming a general three dimensional situation, the term on the left hand side, through some mathematical manipulation [17], can be rewritten as

$$\left(-\boldsymbol{\kappa}^{\phi}\cdot\nabla\phi\right)_f \cdot \mathbf{S}_f = -\begin{bmatrix} \kappa_{11}^{\phi} & \kappa_{12}^{\phi} & \kappa_{13}^{\phi} \\ \kappa_{21}^{\phi} & \kappa_{22}^{\phi} & \kappa_{23}^{\phi} \\ \kappa_{31}^{\phi} & \kappa_{32}^{\phi} & \kappa_{33}^{\phi} \end{bmatrix}_f \begin{bmatrix} \frac{\partial\phi}{\partial x} \\ \frac{\partial\phi}{\partial y} \\ \frac{\partial\phi}{\partial z} \end{bmatrix}_f \cdot \mathbf{S}_f \tag{8.91}$$

Performing matrix multiplication, Eq. (8.91) becomes

$$\left(-\boldsymbol{\kappa}^{\phi}\cdot\nabla\phi\right)_f \cdot \mathbf{S}_f = -\begin{bmatrix} \kappa_{11}^{\phi}\frac{\partial\phi}{\partial x} + \kappa_{12}^{\phi}\frac{\partial\phi}{\partial y} + \kappa_{13}^{\phi}\frac{\partial\phi}{\partial z} \\ \kappa_{21}^{\phi}\frac{\partial\phi}{\partial x} + \kappa_{22}^{\phi}\frac{\partial\phi}{\partial y} + \kappa_{23}^{\phi}\frac{\partial\phi}{\partial z} \\ \kappa_{31}^{\phi}\frac{\partial\phi}{\partial x} + \kappa_{32}^{\phi}\frac{\partial\phi}{\partial y} + \kappa_{33}^{\phi}\frac{\partial\phi}{\partial z} \end{bmatrix}_f \begin{bmatrix} S^x \\ S^y \\ S^z \end{bmatrix}_f \tag{8.92}$$

Further manipulations yield

$$\begin{aligned} \left(-\boldsymbol{\kappa}^{\phi}\cdot\nabla\phi\right)_f \cdot \mathbf{S}_f &= -\begin{bmatrix} \frac{\partial\phi}{\partial x} & \frac{\partial\phi}{\partial y} & \frac{\partial\phi}{\partial z} \end{bmatrix}_f \begin{bmatrix} \kappa_{11}^{\phi} & \kappa_{21}^{\phi} & \kappa_{31}^{\phi} \\ \kappa_{12}^{\phi} & \kappa_{22}^{\phi} & \kappa_{32}^{\phi} \\ \kappa_{13}^{\phi} & \kappa_{23}^{\phi} & \kappa_{33}^{\phi} \end{bmatrix}_f \begin{bmatrix} S^x \\ S^y \\ S^z \end{bmatrix}_f \\ &= -(\nabla\phi)_f \cdot \left[\left(\boldsymbol{\kappa}^{\phi}\right)^T \cdot \mathbf{S}\right]_f = -(\nabla\phi)_f \cdot \mathbf{S}'_f \end{aligned} \tag{8.93}$$

Substituting Eq. (8.93) into Eq. (8.90), the new form of the diffusion equation becomes

$$\sum_{f \sim nb(C)} (-\nabla\phi)_f \cdot \mathbf{S}'_f = Q_C^{\phi} V_C \tag{8.94}$$

It is obvious that in this form the discretization procedure described above can be applied by simply setting Γ^{ϕ} to 1 and replacing $\mathbf{S}_f$ by $\mathbf{S}'_f$. Therefore the same code can be used to solve isotropic and anisotropic diffusion problems.

8.9 Under-Relaxation of the Iterative Solution Process

For a general diffusion problem, Γ^{ϕ} may be a function of the unknown dependent variable ϕ and the grid may be highly non-orthogonal with a large cross diffusion term, which is treated using a deferred correction approach. Therefore large variations in ϕ between iterations result in large source terms and large changes in the coefficients, which may cause divergence of the iterative solution procedure. This divergence is usually due to the non-linearity introduced by the coefficients and the cross-diffusion term, which makes the source term highly affected by the current solution field which is not yet converged. To promote convergence and stabilize the iterative solution process, slowing down the changes in ϕ between iterations is highly desirable and is enforced by a technique called ***under-relaxation***. There are many ways of introducing under-relaxation. One of the practices will be described here, others in Chap. 14. Derivations will be performed on the general discretization equation of the form

$$a_C \phi_C + \sum_{F \sim NB(C)} a_F \phi_F = b_C \tag{8.95}$$

Equation (8.95) can be written as

$$\phi_C = \frac{-\sum_{F \sim NB(C)} a_F \phi_F + b_C}{a_C} \tag{8.96}$$

Let ϕ_C^* represents the value of ϕ_C from the previous iteration. If ϕ_C^* is added to and subtracted from the right hand side, then Eq. (8.96) becomes

$$\phi_C = \phi_C^* + \left(\frac{-\sum_{F \sim NB(C)} a_F \phi_F + b_C}{a_C} - \phi_C^* \right) \tag{8.97}$$

where the expression between the parentheses represents the change in $\boldsymbol{\phi_C}$ produced by the current iteration. This change can be modified by the introduction of a relaxation factor λ^{ϕ}, such that

$$\phi_C = \phi_C^* + \lambda^\phi \left(\frac{-\sum\limits_{F \sim NB(C)} a_F \phi_F + b_C}{a_C} - \phi_C^* \right) \tag{8.98}$$

or

$$\frac{a_C}{\lambda^\phi} \phi_C + \sum_{F \sim NB(C)} a_F \phi_F = b_C + \frac{(1 - \lambda^\phi) a_C}{\lambda^\phi} \phi_C^* \tag{8.99}$$

At first, it should be noted that at convergence ϕ_C and ϕ_C^* become equal independent of the value of relaxation factor used. This is indeed reflected by Eq. (8.98), which shows that the converged value ϕ_C does satisfy the original equation (Eq. 8.95). This property should be satisfied by any relaxation scheme.

Depending on the value of the relaxation factor λ^ϕ, the equation may be either under relaxed (i.e., $0 < \lambda^\phi < 1$) or over-relaxed (i.e., $\lambda^\phi > 1$). In CFD applications, under relaxation is usually used. A value of λ^ϕ close to 1 implies little under relaxation, while a value close to 0 produces heavy under relaxation effects with very small changes in ϕ_C from iteration to iteration.

The optimum under relaxation factor is problem dependent and is not governed by any general rule. The factors affecting λ^ϕ values include the type of problem solved, the size of the system of equation (i.e., number of grid points in the domain), the grid spacing and its expansion rate, and the adopted iterative method, among others. Usually, values of λ^ϕ are assigned based on experience or from preliminary calculations. Moreover, it is not necessary to use the same under-relaxation value throughout the computational domain and values may vary from iteration to iteration.

Equation (8.99) can be recast into the form of Eq. (8.95), where now the central coefficient a_c becomes

$$a_C \leftarrow \frac{a_C}{\lambda^\phi} \tag{8.100}$$

while the source term is added to produce a new term as

$$b_C \leftarrow b_C + \frac{(1 - \lambda^\phi) a_C}{\lambda^\phi} \phi_C^* \tag{8.101}$$

This method of relaxation plays an important role in stabilizing the solution of non-linear problems.

8.10 Computational Pointers

8.10.1 uFVM

In uFVM the implementation of the diffusion term discretization for interior faces is performed in function **cfdAssembleDiffusionTermInterior**, the core of which is shown in Listing 8.1.

```
gamma_f = cfdInterpolateFromElementsToFaces('Average',gamma);
gamma_f = [gamma_f(iFaces)];
geoDiff_f = [theMesh.faces(iFaces).geoDiff]';
Sf = [theMesh.faces(iFaces).Sf]';
Tf = [theMesh.faces(iFaces).T]';

iOwners = [theMesh.faces(iFaces).iOwner]';
iNeighbours = [theMesh.faces(iFaces).iNeighbour]';


theFluxes.FLUXC1f(iFaces,1) =  gamma_f .* gDiff_f;
theFluxes.FLUXC2f(iFaces,1) = -gamma_f .* gDiff_f;

theFluxes.FLUXVf(iFaces,1)  =  gamma_f .* dot(grad_f(:,:)',Tf(:,:)')';
theFluxes.FLUXTf(iFaces,1)       =       theFluxes.FLUXC1f(iFaces)    .*
phi(iOwners)   +   theFluxes.FLUXC2f(iFaces)   .*   phi(iNeighbours)   +
theFluxes.FLUXVf(iFaces);
```

Listing 8.1 Assembly of the diffusion term at interior faces

In the above, **`FLUXC1f`** and **`FLUXC2f`** are equivalent to $FluxC_f$ and $FluxF_f$ in the text and are the coefficients of the owner and neighbor element, respectively. Moreover, **`gamma_f`** and **`gDiff_f`** are arrays defined over all interior faces, and using the dot notation, the expression **`gamma_f.* gDiff_f`** returns an array containing the product of the respective elements of the **`gamma_f`** and **`gDiff_f`** arrays. The dot notation in Matlab® allows for efficient operations over arrays and for the writing of a clearer code and is used in uFVM whenever practical.

The non-orthogonal term is stored in the **`FLUXVf`** coefficient and is equal to **`gamma_f .* dot(grad_f(: , :)',Tf( :, :)')'`**.The total flux passing through face *f*, **`FLUXTf`**, is assembled as in Eq. (8.15).

The diffusion term is setup in **cfdProcessOpenFoamMesh.m** and Listing 8.2 shows the computation of the **`gDiff`** coefficient.

```
theMesh.faces(iFace).dCF = element2.centroid - element1.centroid;
eCF = dCF/cfdMagnitude(dCF);
E = theFace.area*eCF;
theMesh.faces(iFace).gDiff = cfdMagnitude(E)/cfdMagnitude(dCF);
theMesh.faces(iFace).T = theFace.Sf - E;
```

Listing 8.2 Computing the geometric diffusion coefficient

The effects of boundary conditions on the equations of boundary elements should be accounted for and Listing 8.3 shows the assembly of the diffusion term at boundary faces for the case of a Dirichlet boundary condition type.

```
theMesh = cfdGetMesh;
numberOfElements = theMesh.numberOfElements;
numberOfInteriorFaces = theMesh.numberOfInteriorFaces;
%
theBoundary = theMesh.boundaries(iPatch);
numberOfBFaces = theBoundary.numberOfBFaces;

%
iFaceStart = theBoundary.startFace;
iFaceEnd = iFaceStart+numberOfBFaces-1;
iBFaces = iFaceStart:iFaceEnd;
%
iElementStart = numberOfElements+iFaceStart-numberOfInteriorFaces;
iElementEnd = iElementStart+numberOfBFaces-1;
iBElements = iElementStart:iElementEnd;

geodiff = [theMesh.faces(iBFaces).geoDiff]';
Tf = [theMesh.faces(iBFaces).T]';
iOwners = [theMesh.faces(iBFaces).iOwner]';

theFluxes.FLUXC1f(iBFaces) =   gamma(iBElements).*geodiff;
theFluxes.FLUXC2f(iBFaces) = - gamma(iBElements).*geodiff;
theFluxes.FLUXVf(iBFaces)                                  =              -
gamma(iBElements).*dot(grad(iBElements,:)',Tf(:,:)')';
```

Listing 8.3 Assembly of the diffusion term at boundary faces for a Dirichlet Condition

The implementation of other boundary condition types can be reviewed in file **`cfdAssembleDiffusionTerm.m`**.

Once the linearized coefficients are computed for each of the interior and boundary faces, then assembly into the global (sparse) matrix can proceed. This approach is used for the discretization of all flux terms in uFVM and the assembly is performed in the **`cfdAssembleIntoGlobalMatrixFaceFluxes`** function. Listing 8.4 shows the assembly into the sparse LHS matrix and the RHS vector of the coefficients at interior faces.

```
%
% Assemble fluxes of interior faces
%
for iFace = 1:numberOfInteriorFaces
    theFace = theMesh.faces(iFace);
    iOwner          = theFace.iOwner;
    iOwnerNeighbourCoef     = theFace.iOwnerNeighbourCoef;
    iNeighbour      = theFace.iNeighbour;
    iNeighbourOwnerCoef = theFace.iNeighbourOwnerCoef;
    %
    %  assemble fluxes for owner cell
    ac(iOwner) = ac(iOwner)
                 + vf_f(iFace)*theFluxes.FLUXC1f(iFace);
 anb{iOwner}(iOwnerNeighbourCoef) = anb{iOwner}(iOwnerNeighbourCoef)
                 + vf_f(iFace)*theFluxes.FLUXC2f(iFace);
    bc(iOwner) = bc(iOwner)
                 - vf_f(iFace)*theFluxes.FLUXTf(iFace);
    %
    %  assemble fluxes for neighbour cell
    ac(iNeighbour) = ac(iNeighbour)
            - vf_f(iFace)*theFluxes.FLUXC2f(iFace);
    anb{iNeighbour}(iNeighbourOwnerCoef)       =       anb{iNeighbour}
(iNeighbourOwnerCoef)
            - vf_f(iFace)*theFluxes.FLUXC1f(iFace);
    bc(iNeighbour)                          = bc(iNeighbour)
            + vf_f(iFace)*theFluxes.FLUXTf(iFace);
end
```

Listing 8.4 Assembly into LHS spare matrix and RHS array

For each interior face the coefficients are assembled into both the owner and neighbor element equations. For the owner the coefficients are (**`ac(iOwner)`**, **`anb(iOwner)(iOwnerNeighbourCoef)`**, and **`bc(iOwner)`**), while for the neighbor the coefficients become: **`ac(iNeighbour)`**, **`anb(iNeighbour)(iNeighbourOwnerCoef)`**, and **`bc(iNeighbour)`**). The different signs used in the assembly of the two equations is due to the fact that the face surface vector is pointing into the neighbor cell and out of the owner cell.

8.10.2 OpenFOAM®

In OpenFOAM® [18] the diffusion term can be evaluated explicitly using "fvc:: laplacian(gamma, phi)"or implicitly via "fvm::laplacian(gamma,phi)" namespaces and functions. The "fvc::laplacian(gamma,phi)" returns a field in which the laplacian of a generic field ϕ (phi) is evaluated at each cell. The field is added to the right hand side of the system of equations. The "fvm::laplacian(gamma,phi)" returns instead an fvMatrix of coefficients evaluated as per Eq. (8.19), which is added to the left hand side of the system of equations, in addition to a field containing the non-orthogonal terms that is added to the right hand side of the system. The definition of the laplacian operator is located in the directory "$FOAM_SRC/finiteVolume/finiteVolume/laplacianSchemes/gaussLaplacianScheme" in the files "gaussLaplacianScheme.C", "gaussLaplacianScheme.H", and "gaussLaplacianSchemes.C".

The “fvm::laplacian” definition displayed in Listing 8.5 reads

```
template<>
Foam::tmp<Foam::fvMatrix<Foam::Type> >
Foam::fv::gaussLaplacianScheme<Foam::Type, Foam::scalar>::fvmLaplacian
(
    const GeometricField<scalar, fvsPatchField, surfaceMesh>& gamma,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    const fvMesh& mesh = this->mesh();

    GeometricField<scalar, fvsPatchField, surfaceMesh> gammaMagSf
    (
        gamma*mesh.magSf()
    );

    tmp<fvMatrix<Type> > tfvm = fvmLaplacianUncorrected
    (
        gammaMagSf,
        this->tsnGradScheme_().deltaCoeffs(vf),
        vf
    );
    fvMatrix<Type>& fvm = tfvm();

    if (this->tsnGradScheme_().corrected())
    {
        if (mesh.fluxRequired(vf.name()))
        {
            fvm.faceFluxCorrectionPtr() = new
            GeometricField<Type, fvsPatchField, surfaceMesh>
            (
                gammaMagSf*this->tsnGradScheme_().correction(vf)
            );

            fvm.source() -=
                mesh.V()*
                fvc::div
                (
                    *fvm.faceFluxCorrectionPtr()
                )().internalField();
        }
        else
        {
            fvm.source() -=
                mesh.V()*
                fvc::div
                (
                    gammaMagSf*this->tsnGradScheme_().correction(vf)
                )().internalField();
        }
    }
    return tfvm;
}
```

Listing 8.5 Calculation of the Laplacian operator

with the following additional functions (Listing 8.6):

```
template<class Type, class GType>
tmp<fvMatrix<Type> >
gaussLaplacianScheme<Type, GType>::fvmLaplacianUncorrected
(
    const surfaceScalarField& gammaMagSf,
    const surfaceScalarField& deltaCoeffs,
    const GeometricField<Type, fvPatchField, volMesh>& vf
)
{
    tmp<fvMatrix<Type> > tfvm
    (
        new fvMatrix<Type>
        (
            vf,
    deltaCoeffs.dimensions()*gammaMagSf.dimensions()*vf.dimensions()
        )
    );
    fvMatrix<Type>& fvm = tfvm();

    fvm.upper()= deltaCoeffs.internalField()*gammaMagSf.internalField();
    fvm.negSumDiag();

    forAll(vf.boundaryField(), patchi)
    {
        const fvPatchField<Type>& pvf = vf.boundaryField()[patchi];
        const  fvsPatchScalarField&  pGamma  =  gammaMagSf.boundaryField()
[patchi];
        const fvsPatchScalarField& pDeltaCoeffs =
            deltaCoeffs.boundaryField()[patchi];

        if (pvf.coupled())
        {
           fvm.internalCoeffs()[patchi] =
               pGamma*pvf.gradientInternalCoeffs(pDeltaCoeffs);

           fvm.boundaryCoeffs()[patchi] =
              -pGamma*pvf.gradientBoundaryCoeffs(pDeltaCoeffs);

        }
        else
        {
           fvm.internalCoeffs()[patchi] = pGamma*pvf.gradientInternalCoeffs();
           fvm.boundaryCoeffs()[patchi] = -pGamma*pvf.gradientBoundaryCoeffs();
        }
    }
    return tfvm;
}
```

Listing 8.6 Additional functions needed for the calculation of the Laplacian operator

It is worth noting that in OpenFOAM® the assembly takes place directly into the global coefficients, which as described earlier in Chaps. 5, 6, and 7 are stored in three arrays, namely **`fvm.upper()`**, **`fvm.lower()`**, and **`fvm.diag()`**. The main part of the discretization is defined in the **`fvmLaplacianUncorrected`** function, where Eq. (8.19) is evaluated by first defining an **`fvMatrix`** object and then filling its upper triangle by the extra diagonal coefficients (this approach relies on the fact that the Laplacian operator returns a symmetric matrix).

The "**deltaCoeffs.internalField()**" represents the ***gDiff*** field while gamma indicates the diffusion field. The diagonal coefficients are evaluated in **fvm.negSumDiag()**,where the negative sum of the upper and lower coefficients are assembled into the diagonal coefficients as per Eq. (8.18).

The "**fvm.negSumDiag()**" method is defined in **lduMatrix Operations.C** as (Listing 8.7)

```
void Foam::lduMatrix::negSumDiag()
{
    const scalarField& Lower = const_cast<const lduMatrix&>(*this).lower();
    const scalarField& Upper = const_cast<const lduMatrix&>(*this).upper();
    scalarField& Diag = diag();

    const labelUList& l = lduAddr().lowerAddr();
    const labelUList& u = lduAddr().upperAddr();

    for (register label face=0; face<l.size(); face++)
    {
        Diag[l[face]] -= Lower[face];
        Diag[u[face]] -= Upper[face];
    }
}
```

Listing 8.7 Calculation of the negative sum of the upper and lower coefficients

The boundary conditions are implemented exactly as in Eqs. (8.36) and (8.41). In OpenFOAM® the boundary coefficients are stored in "**internalCoeffs**" ($FluxC_b$) and "**boundaryCoeffs**" ($\boldsymbol{FluxV_b}$), already defined in Sect. 7.6.

In "**fvmUncorrected**"only the orthogonal discretization is accounted for. The non-orthogonal contribution is added, as shown in Listing 8.8, using

```
fvm.faceFluxCorrectionPtr() = new
 GeometricField<Type, fvsPatchField, surfaceMesh>
 (
     gammaMagSf*this->tsnGradScheme_().correction(vf)
 );

 fvm.source() -=
     mesh.V()*
     fvc::div
     (
         *fvm.faceFluxCorrectionPtr()
     )().internalField();
```

Listing 8.8 Adding the nonorthogonal diffusion contribution

Again this term represents exactly the implementation of the last term of Eq. (8.77) in which the *snGrad* class wraps into the correction function the non-orthogonal term as shown below in Listing (8.9).

```
template<class Type>
Foam::tmp<Foam::GeometricField<Type,
Foam::fvsPatchField,
Foam::surfaceMesh> >
Foam::fv::correctedSnGrad<Type>::correction
(
    const GeometricField<Type, fvPatchField, volMesh>& vf
) const
{
    const fvMesh& mesh = this->mesh();

    // construct GeometricField<Type, fvsPatchField, surfaceMesh>
    tmp<GeometricField<Type, fvsPatchField, surfaceMesh> > tssf
    (
        new GeometricField<Type, fvsPatchField, surfaceMesh>
        (
             IOobject
            (
                "snGradCorr("+vf.name()+')',
                vf.instance(),
                mesh,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            mesh,
            vf.dimensions()*mesh.nonOrthDeltaCoeffs().dimensions()
        )
    );
    GeometricField<Type, fvsPatchField, surfaceMesh>& ssf = tssf();

    for (direction cmpt = 0; cmpt < pTraits<Type>::nComponents; cmpt++)
    {
        ssf.replace
        (
            cmpt,
            correctedSnGrad<typename pTraits<Type>::cmptType>(mesh)
           .fullGradCorrection(vf.component(cmpt))
        );
    }

template<class Type>
Foam::tmp<Foam::GeometricField<Type,  Foam::fvsPatchField,  Foam::surfaceMesh>
>Foam::fv::correctedSnGrad<Type>::fullGradCorrection
(
    const GeometricField<Type, fvPatchField, volMesh>& vf
) const
{
    const fvMesh& mesh = this->mesh();

    // construct GeometricField<Type, fvsPatchField, surfaceMesh>
    tmp<GeometricField<Type, fvsPatchField, surfaceMesh> > tssf =
        mesh.nonOrthCorrectionVectors()
      & linear<typename outerProduct<vector, Type>::type>(mesh).interpolate
        (
            gradScheme<Type>::New
            (
                mesh,
                mesh.gradScheme("grad(" + vf.name() + ')')
            )().grad(vf, "grad(" + vf.name() + ')')
        );
    tssf().rename("snGradCorr(" + vf.name() + ')');

    return tssf;
}
```

Listing 8.9 Implementation of the nonorthogonal term into the correction function

The non-orthogonal correction term is defined in the *fullGradCorrection* function where "mesh.nonOrthCorrectionVectors()" returns the **T** vector of Eq. (8.66).

The type of Laplacian discretization to be used is specified in the "fvSchemes" file, which is part of the case definition and is located in the system directory

```
laplacianSchemes
{
    laplacian(gamma,phi) Gauss linear corrected;
}
```

Listing 8.10 The discretization type for the Laplacian operator

(Listing 8.10), in which **Gauss** (the only available choice) defines the standard **Gauss** discretization resulting in Eq. (8.18), **linear** refers to the type of interpolation used for calculating the diffusivity gamma at the face, and **corrected** describes the kind of non-orthogonal correction.

More details on the implementation of boundary conditions in OpenFOAM® are presented in later chapters.

8.11 Closure

In this chapter the discretization of the diffusion equation was described. A number of issues were also addressed, such as the use of orthogonal and non-orthogonal grid systems, the implementation of boundary conditions, and under and over relaxation. The next chapter will concentrate on the calculation of the gradient field.

8.12 Exercises

Exercise I

Consider the diffusion of a property ϕ in the one-dimensional domain shown in Fig. 8.16 with no internal sources. The domain is subdivided into 5 uniform elements of size $\Delta x = 1$ and subject to a Dirichlet and a Neumann condition at boundary '0' and '6', respectively. The diffusion coefficient in the domain is constant with a value of 1, i.e., $\Gamma^{\phi} = 1$.

a. Derive the discrete equation for each of the elements.
b. Solve the system of equations using the Gauss-Seidel iterative method and report the resulting cell-centroid values.
c. Compute the diffusion flux $(-\Gamma d\phi/dx)$ at each of the cell faces and show that conservation is satisfied throughout the domain

d. Compare your solution with the exact solution (Note that even though the computed solution is *conservative*, it is not *exact*).
e. For the case where a zero flux boundary condition is defined at both boundaries '0' and '6', reformulate the equations for elements 1 and 5 and explain why the equations cannot be solved.

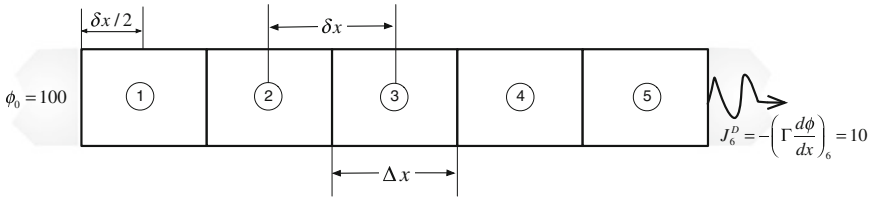

Fig. 8.16 Computational domain for a one dimensional diffusion problem with no internal sources

Exercise 2

A fin is exposed to the surrounding fluid at a temperature T_a = 25 °C, as shown in Fig. 8.17, with a heat transfer coefficient of h = 100 W/m^2 K and a thermal conductivity with value of k = 160 W/m K. The fin has a length L = 0.1 m, a cross sectional area $A = 10^{-5}$ m^2, and a perimeter P = 0.1004 m.

The temperature distribution in the fin is governed by the following differential equation

$$-\frac{d}{dx}\left(k\frac{dT}{dx}\right)+\frac{Ph}{A}(T-T_a)=0$$

Discretize the above equation by subdividing the computational domain into 5 elements of equal size and find the values of the temperature field at the element centroids and boundaries in the following two situations:

a. $T(x=0)=200\ ^\circ\text{C}$ and $T(x=L)=90\ ^\circ\text{C}$
b. $q(x=0)=20\ \text{kW/m}^2$ and $q(x=L)=10\ \text{kW/m}^2$

where q is the rate of heat transfer given by

$$q=-k\frac{dT}{dx}$$

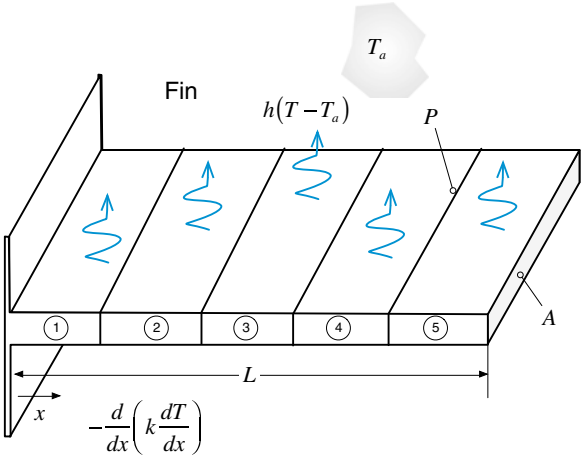

Fig. 8.17 Computational domain for heat transfer from a fin

Exercise 3

The heat conduction in the two-dimensional domain shown in Fig. 8.18 is governed by the following differential equation:

$$-\nabla \cdot k\nabla T = 0$$

The domain is subdivided into uniform elements and the boundary conditions are as shown in the figure.

a. Derive the algebraic equations for all elements.
b. Solve the system of equations obtained and compute the T values at the centroids of the elements.

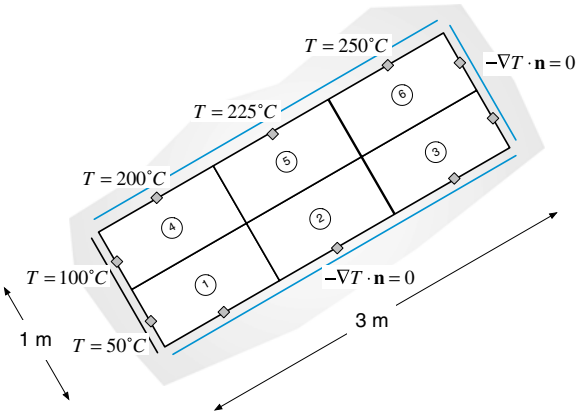

Fig. 8.18 Heat conduction in a two dimensional tilted Cartesian domain

c. Compute the values of T at the bottom and right boundaries.
d. Compute the net heat transfer through the top and left boundaries

Exercise 4

Consider steady state conduction heat transfer in the non-orthogonal domain discretized into the four equal elements shown in Fig. 8.19, where $L = 1$ m. Derive the discretization equations for the cells using two different methods for the decomposition of the diffusion flux and compare the temperature values at the element centroids after 4 coefficient iterations.

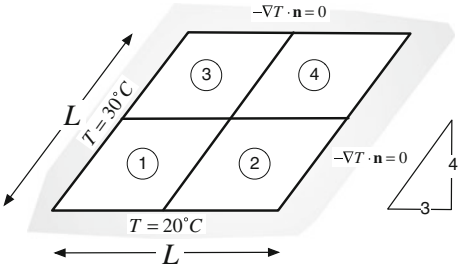

Fig. 8.19 Computational domain for Exercise 4

Exercise 5

Discretize the equation $-\nabla \cdot k\nabla T = Q^T$ where $Q^T = 500$ and $k = 200$, for the mesh composed of quadrilateral triangles of side 0.1 shown in Fig. 8.20. Write the discretized equations in the form of general algebraic equations.

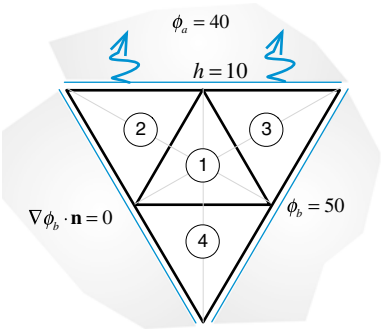

Fig. 8.20 A triangular domain covered with an unstructured grid system

Exercise 6
The gradient for a variable ϕ over the computational domain shown in Fig. 8.21 is given by

$$\nabla\phi = 20\mathbf{i} + 30\mathbf{j}$$

Compute the value of ϕ at nodes 1 and 2 given that the length and width of the computational domain are 10 and 5 cm, respectively.

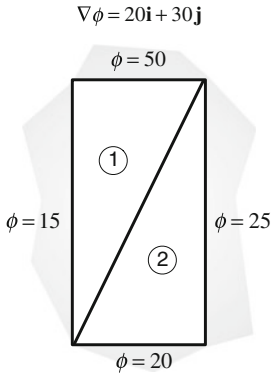

Fig. 8.21 A rectangular domain decomposed into two triangular elements

Exercise 7
Using the mesh constructed in exercise 3 of Chap. 7 (displayed in Fig. 7.12), setup a case in OpenFOAM® and uFVM to solve the diffusion equation subject to the following conditions ($\Gamma^{\phi} = 1$):
Patch#1 $\phi = 1$
Patch#2 $\phi = 0$
Patch#3 $\nabla\phi \cdot \mathbf{n} = 0$
Patch#4 $h_{\infty} = 1$, $\phi_{\infty} = 0.5$

Exercise 8
Build an oblique parallelogram (size of horizontal side is 1 and size of oblique side is 2 inclined at 60 degrees with respect to the horizontal) using blockMesh in OpenFOAM® with the boundary conditions shown in the figure below to solve the diffusion equation with no source term in two dimensions. Generate a grid by decomposing each side of the parallelogram into 50 equal segments. Setup the case and using a diffusion coefficient of 1 compare the convergence history for the three different approaches mentioned in this chapter to resolve the non-orthogonal term for the diffusion equation (one at a time) in uFVM and in OpenFOAM®. Use an under-relaxation factor of value 0.9 (Fig. 8.22).

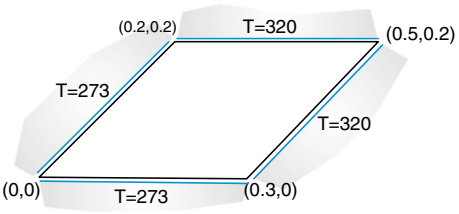

Fig. 8.22 An oblique parallelogram

Exercise 9

a. Use Doxygen [19] to find the list of all overloaded functions under the name fvmLaplacian and then under then name fvcLaplacian.
b. Define in the dictionary file fvSchemes the option to exclude the non orthogonal correction (Gauss linear uncorrected;).
c. Define in the dictionary file fvSchemes the option to use a limited non orthogonal correction (Gauss linear limited 0.8;).
d. List of the possible Laplacian non orthogonal correction type available in OpenFOAM® (Hint: just mistype a scheme, i.e., banana, and launch any solver or application).

References

1. Patankar SV (1980) Numerical heat transfer and fluid flow. Hemisphere Publishing Corporation, McGraw-Hill, USA
2. Jiang T, Przekwas AJ (1994) Implicit, pressure-based incompressible Navier-stokes equations solver for unstructured meshes. AIAA-94-0305
3. Davidson L (1996) A pressure correction method for unstructured meshes with arbitrary control volumes. Int J Numer Meth Fluids 22:265–281
4. Barth TJ (1992) Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-stokes equations. Special course on unstructured grid methods for advection dominated flows. AGARD Report 787
5. Perez-Segarra CD, Farre C, Cadafalch J, Oliva A (2006) Analysis of different numerical schemes for the resolution of convection-diffusion equations using finite-volume methods on three-dimensional unstructured grids, Part I: discretization schemes. Numer Heat Transfer Part B: Fundam 49(4):333–350
6. Demirdzic I, Lilek Z, Peric M (1990) Finite volume method for prediction of fluid flow in arbitrary shaped domains with moving boundary. Int J Numer Meth Fluids 10:771–790
7. Demirdzic I, Lilek Z, Peric M (1993) A collocated finite volume method for predicting flows at all speeds. Int J Numer Meth Fluids 16:1029–1050
8. Warsi ZUA (1993) Fluid dynamics: theoretical and computational approaches. CRC Press, Boca Raton
9. Berend FD, van Wachem GM (2014) Compressive VOF method with skewness correction to capture sharp interfaces on arbitrary meshes. J Comput Phys 279:127–144
10. Jasak H (1996) Error analysis and estimation for the finite volume method with applications to fluid flow. Ph.D. thesis, Imperial College London

11. Balckwell BF, Hogan RE (1993) Numerical solution of axisymmetric heat conduction problems using the finite control volume technique. J Thermophys Heat Transfer 7:462–471
12. Wang S (2002) Solving convection-dominated anisotropic diffusion equations by an exponentially fitted finite volume method. Comput Math Appl 44:1249–1265
13. Jayantha PA, Turner IW (2003) On the use of surface interpolation techniques in generalised finite volume strategies for simulating transport in highly anisotropic porous media. J Comput Appl Math 152:199–216
14. Bertolazzi E, Manzini G (2006) A second-order maximum principle preserving finite volume method for steady convection-diffusion problems. SIAM J Numer Anal 43:2172–2199
15. Domelevo K, Omnes P (2005) A finite volume method for the Laplace equation on almost arbitrary two-dimensional grids. Math Model Numer Anal 39:1203–1249
16. Eymard E, Gallouet T, Herbin R (2004) A finite volume scheme for anisotropic diffusion problems. Comptes Rendus Mathematiques (CR MATH) 339:299–302
17. Darwish M, Moukalled F (2009) A compact procedure for discretization of the anisotropic diffusion operator. Numer Heat Transfer, Part B 55:339–360
18. OpenFOAM, 2015 Version 2.3.x. http://www.openfoam.org
19. OpenFOAM Doxygen, 2015 Version 2.3.x. http://www.openfoam.org/docs/cpp/